THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Advanced Automata Minimization

Citation for published version:
Mayr, R & Clemente, L 2013, Advanced Automata Minimization. in *POPL '13 Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages.* ACM, pp. 63-74. https://doi.org/10.1145/2429069.2429079

Digital Object Identifier (DOI):
10.1145/2429069.2429079

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Early version, also known as pre-print

Published In:
POPL '13 Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 07. May. 2023

Advanced Automata Minimization

Lorenzo Clemente
LaBRI, University of Bordeaux I
lorenzo.clemente@labri.fr

Richard Mayr
University of Edinburgh
http://homepages.inf.ed.ac.uk/rmayr

arXiv:1210.6624v1 [cs.FL] 24 Oct 2012

Abstract

We present an efficient algorithm to reduce the size of nondeterministic Büchi word automata, while retaining their language. Additionally, we describe methods to solve PSPACE-complete automata problems like universality, equivalence and inclusion for much larger instances (1-3 orders of magnitude) than before. This can be used to scale up applications of automata in formal verification tools and decision procedures for logical theories.

The algorithm is based on new transition pruning techniques. These use criteria based on combinations of backward and forward trace inclusions. Since these relations are themselves PSPACE-complete, we describe methods to compute good approximations of them in polynomial time.

Extensive experiments show that the average-case complexity of our algorithm scales quadratically. The size reduction of the automata depends very much on the class of instances, but our algorithm consistently outperforms all previous techniques by a wide margin. We tested our algorithm on Büchi automata derived from LTL-formulae, many classes of random automata and automata derived from mutual exclusion protocols, and compared its performance to the well-known automata tool GOAL [34].

Categories and Subject Descriptors D.2.4 [*Software Verification*]: Model checking; F.1.1 [*Models of Computation*]: Automata

General Terms Automata minimization, inclusion checking

Keywords Büchi automata, simulation, minimization

1. Introduction

Nondeterministic Büchi automata are an effective way to represent and manipulate ω-regular languages, since they are closed under boolean operations. They appear in many automata-based formal software verification methods, as well as in decision procedures for logical theories. For example, in LTL software model checking [13, 22], temporal logic specifications are converted into Büchi automata. In other cases, different versions of a program (obtained by abstraction or refinement of the original) are translated into automata whose languages are then compared. Testing the conformance of an implementation with its requirements specification thus reduces to a language inclusion or language equivalence problem. Another application of Büchi automata in software engineering is program termination analysis by the size-change termination method [15, 26]. Via an abstraction of the effect of program operations on data, the termination problem can often be reduced to a language inclusion problem about two derived Büchi automata.

Our goal is to improve the efficiency and scalability of automata-based formal software verification methods. We consider efficient algorithms for the minimization of automata, in the sense of obtaining a smaller automaton with the same language, though not necessarily with the absolute minimal possible number of states. (And, in general, the minimal automaton for a language is not even unique.) The reason to perform minimization is that the smaller minimized automaton is more efficient to handle in a subsequent computation. Thus there is an algorithmic tradeoff between the effort for minimization and the complexity of the problem later considered for this automaton. If only computationally easy questions are asked (e.g., reachability/emptiness; solvable in Logspace/PTIME) then extensive minimization usually does not pay off. Instead, the main applications are the following:

1. Computationally hard automata problems like universality, equivalence, and inclusion. These are PSPACE-complete [25], but many practically efficient methods have been developed [3, 4, 6, 10, 11, 15, 16, 29]. Still, these all have exponential time complexity and do not scale well. Typically they are applied to automata with 15–100 states (unless the automaton has a particularly simple structure). Thus, one should first minimize the automata before applying these exponential-time methods. A good minimization algorithm makes it possible to solve much larger instances. Even better, many instances of the PSPACE-complete universality, equivalence, and inclusion problems can already be solved in the *polynomial time* minimization algorithm (e.g., by reducing the automaton to the trivial universal automaton), so that the complete *exponential time* methods only need to be invoked in a small minority of instances.
2. Cases where the size of an automaton strongly affects the complexity of an algorithm. In LTL model checking [22] one searches for loops in a graph that is the *product* of a large system specification with an automaton derived from an LTL-formula. Smaller automata often make this easier, though in practice it also depends on the degree of nondeterminism [30].
3. Procedures that combine and modify automata repeatedly. Model checking algorithms and automata-based decision procedures for logical theories compute automata products, unions, complements, projections, etc., and thus the sizes of automata grow rapidly. Thus, it is important to intermittently minimize the automata to keep their size manageable, e.g., [27].

In general, finding an automaton with the minimal number of states for a given language is computationally hard; even deciding whether a given automaton is minimal is already PSPACE-complete [23]. Thus much effort has been devoted to finding methods for partial minimization [8, 13, 14, 24]. Simulation preorders played a central role in these efforts, because they provide PTIME-

[Technical Report EDI-INF-RR-1414 of the School of Informatics at the University of Edinburgh, UK. http://www.inf.ed.ac.uk/publications/report/
Made available at arXiv.org - Non-exclusive license to distribute (http://arxiv.org/licenses/nonexclusive-distrib/1.0/).

Research supported by UK Royal Society Grant IE110996.]

computable under-approximations of trace inclusions. However, the quality of the approximation is insufficient in many practical examples. Multipebble simulations [12] yield coarser relations by allowing the Duplicator player to hedge her bets in the simulation game, but they are not easily computable in practice.

1. We present methods for transition pruning, i.e., removing transitions from automata without changing their language. The idea is that certain transitions can be removed, because other 'better' transitions remain. The 'better' criterion relies on combinations of forward and backward simulations and trace inclusions. We provide a complete picture which combinations are correct to use for pruning. Moreover, the pruned transitions can be removed 'in parallel' (i.e., without re-computing the simulations and trace inclusions after every change), which makes the method efficient and practical.
2. We present an efficient practical method to compute good under-approximations of trace inclusions, by introducing *lookahead simulations*. While it is correct to use full trace inclusions and maximal-pebble multipebble simulations in our minimization methods, these are not easily computed (PSPACE-hard). However, lookahead simulations are PTIME-computable, and it is correct to use them instead of the more expensive trace inclusions and multipebble simulations. Lookahead itself is a classic concept in parsing and many other areas, but it can be defined in many different variants. Our contribution is to identify and formally describe the lookahead-variant for simulation preorders that gives the optimal compromise between efficient computability and maximizing the sizes of the relations.[1] Practical degrees of lookahead range from 4 to 25, depending on the size and shape of the automata. Our experiments show that even moderate lookahead helps considerably in obtaining good approximations of trace-inclusions and multipebble simulations.
3. We show that variants of the *polynomial time* minimization algorithm can solve most instances of the PSPACE-complete language inclusion problem. Thus, the complete *exponential time* methods of [3, 4, 6, 10, 11, 15, 16] need only be invoked in a minority of the cases. This allows to scale language inclusion testing to much larger instances (e.g., automata with ≥ 1000 states) which are beyond traditional methods.
4. We performed extensive tests of our algorithm on automata of up-to 20000 states. These included random automata according to the Tabakov-Vardi model [33], automata obtained from LTL formulae, and real-world mutual exclusion protocols. The empirically determined average-case time complexity on random automata is quadratic, while the (never observed) worst-case complexity is $O(n^4)$. The worst-case space complexity is quadratic. Our algorithm always minimizes better, on average, than all previously available practical methods. However, the exact advantage varies, depending on the type of instances; cf. Section 7. For example, consider random automata with 100–1000 states, binary alphabet and varying transition density *td*. Random automata with $td = 1.4$ cannot be minimized much by *any method*. The only effect is achieved by the trivial removal of dead states which, on average, yields automata of 78% of the original size. On the other hand, for $td = 1.8, \ldots, 2.2$, the best previous minimization methods yielded automata of 85%–90% of the original size on average, while our algorithm yielded automata of 3%–15% of the original size on average.

While we present our methods in the framework of Büchi automata, they directly carry over to the simpler case of finite-word automata.

[1] A thorough literature search showed that this has never been formally described so far.

2. Preliminaries

A *non-deterministic Büchi Automaton (BA)* $\mathcal{A}$ is a tuple $(\Sigma, Q, I, F, \delta)$ where Σ is a finite alphabet, Q is a finite set of states, $I \subseteq Q$ is the set of *initial* states, $F \subseteq Q$ is the set of *accepting* states, and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation. We write $p \xrightarrow{\sigma} q$ for $(p, \sigma, q) \in \delta$. A transition is *transient* iff any path can contain it at most once. To simplify the presentation, we assume that automata are *forward and backward complete*, i.e., for any state $p \in Q$ and symbol $\sigma \in \Sigma$, there exist states $q_0, q_1 \in Q$ s.t. $q_0 \xrightarrow{\sigma} p \xrightarrow{\sigma} q_1$. Every automaton can be converted into an equivalent complete one by adding at most two states and a linear number of transitions.[2] A state is *dead* iff either it is not reachable from an initial state, or it cannot reach an accepting loop. In our simplification techniques, we always remove dead states.

A Büchi automaton $\mathcal{A}$ describes a set of infinite words (its language), i.e., a subset of Σ^ω. An *infinite trace* of $\mathcal{A}$ on a word $w = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$ (or *w-trace*) *starting* in a state $q_0 \in Q$ is an infinite sequence of transitions $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$. By $\pi[0..i]$ we denote the finite prefix $\pi = q_0 \xrightarrow{\sigma_0} \cdots \xrightarrow{\sigma_{i-1}} q_i$, and by $\pi[i..]$ the infinite suffix $q_i \xrightarrow{\sigma_i} q_{i+1} \xrightarrow{\sigma_{i+1}} \cdots$. Finite traces starting in q_0 and *ending* in a state $q_m \in Q$ are defined similarly. A finite or infinite trace is *initial* iff it starts in an initial state $q_0 \in I$; if it is infinite, then it is *fair* iff $q_i \in F$ for infinitely many i. The *language of* $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^\omega \mid \mathcal{A} \text{ has an infinite, initial and fair trace on } w\}$.

Language inclusion. When automata are viewed as a finite representation for languages, it is natural to ask whether two different automata represent the same language, or, more generally, to compare these languages for inclusion. Formally, for two automata $\mathcal{A} = (\Sigma, Q_\mathcal{A}, I_\mathcal{A}, F_\mathcal{A}, \delta_\mathcal{A})$ and $\mathcal{B} = (\Sigma, Q_\mathcal{B}, I_\mathcal{B}, F_\mathcal{B}, \delta_\mathcal{B})$ we write $\mathcal{A} \subseteq \mathcal{B}$ iff $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ and $\mathcal{A} \approx \mathcal{B}$ iff $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. The *language inclusion/equivalence problem* consists in determining whether $\mathcal{A} \subseteq \mathcal{B}$ or $\mathcal{A} \approx \mathcal{B}$ holds, respectively. For general non-deterministic automata, language inclusion and equivalence are PSPACE-complete [25] (which entails that, under standard theoretic-complexity assumptions, they admit no efficient deterministic algorithm). Therefore, one considers suitable under-approximations.

Definition A preorder $\sqsubseteq$ on $Q_\mathcal{A} \times Q_\mathcal{B}$ is *good for inclusion* (GFI) iff the following holds: If $\forall q \in I_\mathcal{A} \exists q' \in I_\mathcal{B} \cdot q \sqsubseteq q'$, then $\mathcal{A} \subseteq \mathcal{B}$.

In other words, GFI preorders give a sufficient condition for inclusion, by matching initial states of $\mathcal{A}$ with initial states of $\mathcal{B}$. (They are not necessary for inclusion since there are several initial states.) Moreover, if computing a GFI preorder is efficient, than also inclusion can be established efficiently. Finally, if a preorder is GFI, then all smaller preorders are GFI too, i.e., GFI is $\subseteq$-downward closed.

Quotienting. Another interesting problem is how to simplify an automaton while preserving its semantics, i.e., its language. Generally, one tries to reduce the number of states/transitions. This is useful because the complexity of decision procedures usually depends on the size of the input automata.

A classical operation for reducing the number of states of an automaton is that of quotienting, where states of the automaton are identified according to a given equivalence, and transitions are projected accordingly. Since in practice we obtain quotienting equivalences from suitable preorders, we directly define quotienting w.r.t. a preorder. Formally, fix a BA $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ and a preorder $\sqsubseteq$ on Q, with induced equivalence $\equiv = \sqsubseteq \cap \sqsupseteq$. Given a state $q \in Q$, we denote by $[q]$ its equivalence class w.r.t. $\equiv$, and, for a set of states $P \subseteq Q$, $[P]$ is the set of equivalence classes $[P] = \{[p] \mid p \in P\}$.

[2] For efficiency reasons, our implementation works directly on incomplete automata.

Definition The *quotient* of $\mathcal{A}$ by $\sqsubseteq$ is $\mathcal{A}/\sqsubseteq = (\Sigma, [Q], [I], [F], \delta')$, where $\delta' = \{([q_1], \sigma, [q_2]) \mid \exists q_1' \in [q_1], q_2' \in [q_2].\, (q_1', \sigma, q_2') \in \delta\}$, i.e., transitions are induced element-wise.

Clearly, every trace $q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ in $\mathcal{A}$ immediately induces a corresponding trace $[q_0] \xrightarrow{\sigma_0} [q_1] \xrightarrow{\sigma_1} \cdots$ in $\mathcal{A}/\sqsubseteq$, which is fair/initial if the former is fair/initial, respectively. Consequently, $\mathcal{A} \subseteq \mathcal{A}/\sqsubseteq$ for *any* preorder $\sqsubseteq$. If, additionally, $\mathcal{A}/\sqsubseteq \subseteq \mathcal{A}$, then we say that the preorder $\sqsubseteq$ is good for quotienting (GFQ).

Definition A preorder $\sqsubseteq$ is *good for quotienting* iff $\mathcal{A}/\sqsubseteq \approx \mathcal{A}$.

Like GFI preorders, also GFQ preorders are downward closed (since a smaller preorder is quotienting "less"). Therefore, we are interested in efficiently computable GFI/GFQ preorders. A classical example is given by *simulation relations*.

Simulation relations. Basic forward simulation is a binary relation on the states of $\mathcal{A}$; it relates states whose behaviors are stepwise related, which allows one to reason about the internal structure of automaton $\mathcal{A}$— i.e., *how* a word is accepted, and not just *whether* it is accepted. Formally, simulation between two states p_0 and q_0 can be described in terms of a game between two players, Spoiler and Duplicator, where the latter wants to prove that q_0 can stepwise mimic any behavior of p_0, and the former wants to disprove it. The game starts in the initial configuration (p_0, q_0). Inductively, given a game configuration (p_i, q_i) at the i-th round of the game, Spoiler chooses a symbol $\sigma_i \in \Sigma$ and a transition $p_i \xrightarrow{\sigma_i} p_{i+1}$. Then, Duplicator responds by choosing a matching transition $q_i \xrightarrow{\sigma_i} q_{i+1}$, and the next configuration is (p_{i+1}, q_{i+1}). Since the automaton is assumed to be complete, the game goes on forever, and the two players build two infinite traces $\pi_0 = p_0 \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_1} \cdots$ and $\pi_1 = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$. The winning condition depends on the type of simulation, and different types have been considered depending on whether one is interested in GFQ or GFI relations. Here, we consider *direct* [10], *delayed* [14] and *fair simulation* [21]. Let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$. Duplicator wins the play if $\mathcal{C}^x(\pi_0, \pi_1)$ holds, where

$$\mathcal{C}^{\mathrm{di}}(\pi_0, \pi_1) \iff \forall (i \geq 0) \cdot p_i \in F \implies q_i \in F \tag{1}$$

$$\mathcal{C}^{\mathrm{de}}(\pi_0, \pi_1) \iff \forall (i \geq 0) \cdot p_i \in F \implies \exists (j \geq i) \cdot q_j \in F \tag{2}$$

$$\mathcal{C}^{\mathrm{f}}(\pi_0, \pi_1) \iff \text{if } \pi_0 \text{ is fair, then } \pi_1 \text{ is fair} \tag{3}$$

Intuitively, direct simulation requires that accepting states are matched immediately (the strongest condition), while in delayed simulation Duplicator is allowed to accept only after a finite delay. In fair simulation (the weakest condition), Duplicator must visit accepting states only if Spoiler visits infinitely many of them. Thus, $\mathcal{C}^{\mathrm{di}}(\pi_0, \pi_1)$ implies $\mathcal{C}^{\mathrm{de}}(\pi_0, \pi_1)$, which, in turn, implies $\mathcal{C}^{\mathrm{f}}(\pi_0, \pi_1)$.

We define x-simulation relation $\sqsubseteq^x \subseteq Q \times Q$ by stipulating that $p_0 \sqsubseteq^x q_0$ iff Duplicator has a winning strategy in the x-simulation game, starting from configuration (p_0, q_0); clearly, $\sqsubseteq^{\mathrm{di}} \subseteq \sqsubseteq^{\mathrm{de}} \subseteq \sqsubseteq^{\mathrm{f}}$. Simulation between states in different automata $\mathcal{A}$ and $\mathcal{B}$ can be computed as a simulation on their disjoint union. All these simulation relations are GFI preorders which can be computed in polynomial time [10, 14, 20]; moreover, direct and delayed simulation are GFQ [14], but fair simulation is not [21].

Lemma 2.1 ([10, 14, 20, 21])**.** *For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, x-simulation $\sqsubseteq^x$ is a PTIME, GFI preorder, and, for $y \in \{\mathrm{di}, \mathrm{de}\}$, $\sqsubseteq^y$ is also GFQ.*

Trace inclusions. While simulations are efficiently computable, their use is often limited by their size, which can be much smaller than other GFI/GFQ preorders. One such example of coarser GFI/GFQ preorders is given by *trace inclusions*, which are obtained through a modification of the simulation game, as follows.

In simulation games, the players build two paths π_0, π_1 by choosing single transitions in an alternating fashion; Duplicator moves by knowing only the next 1-step move of Spoiler. We can obtain coarser relations by allowing Duplicator a certain amount of *lookahead* on Spoiler's moves. In the extremal case of ω-lookahead, i.e., where Spoiler has to reveal her whole path in advance, we obtain trace inclusions.

Analogously to simulations, we define direct, delayed, and fair trace inclusion, as binary relations on Q. For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, *$x$-trace inclusion* holds between p and q, written $p \subseteq^x q$ iff, for every word $w = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$, and for every infinite w-trace $\pi_0 = p_0 \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_1} \cdots$ starting at $p_0 = p$, there exists an infinite w-trace $\pi_1 = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ starting at $q_0 = q$, s.t. $\mathcal{C}^x(\pi_0, \pi_1)$. All these trace inclusions are GFI preorders subsuming the corresponding simulation, i.e., $\sqsubseteq^x \subseteq \subseteq^x$ (since Duplicator has more power in the trace inclusion game); also, $\subseteq^{\mathrm{di}}$ is a subset of $\subseteq^{\mathrm{de}}$, which, in turn, is a subset of $\subseteq^{\mathrm{f}}$. Regarding quotienting, $\subseteq^{\mathrm{di}}$ is GFQ (like $\sqsubseteq^{\mathrm{di}}$; this follows from [12]), while $\subseteq^{\mathrm{f}}$ is not, since it is coarser than fair simulation, which is not GFQ [21]. While delayed simulation $\sqsubseteq^{\mathrm{de}}$ is GFQ, delayed trace inclusion $\subseteq^{\mathrm{de}}$ is not GFQ [8].

Lemma 2.2. *For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}\}$, x-trace inclusion $\subseteq^x$ is a GFI preorder. Moreover, $\subseteq^{\mathrm{di}}$ is a GFQ preorder.*

Finally, though $\sqsubseteq^{\mathrm{de}}$ and $\subseteq^{\mathrm{di}}$ are incomparable, there exists a common generalization included in $\subseteq^{\mathrm{de}}$ called *delayed fixed-word simulation* which is GFQ [8].[3]

Backward simulation and trace inclusion. Yet another way of obtaining GFQ/GFI preorders is to consider variants of simulation/trace inclusion which go backwards in time. *Backward simulation* $\sqsubseteq^{\mathrm{bw}}$ ([32], where it is called *reverse simulation*) is defined like ordinary simulation, except that transitions are taken backwards: From configuration (p_i, q_i), Spoiler selects a transition $p_{i+1} \xrightarrow{\sigma_i} p_i$, Duplicator replies with a transition $q_{i+1} \xrightarrow{\sigma_i} q_i$, and the next configuration is (p_{i+1}, q_{i+1}). Let π_0 and π_1 be the two infinite backward traces built in this way. The corresponding winning condition considers both accepting and initial states:

$$\mathcal{C}^{\mathrm{bw}}(\pi_0, \pi_1) \iff \forall (i \geq 0) \cdot \begin{cases} p_i \in F \implies q_i \in F, \text{ and} \\ p_i \in I \implies q_i \in I \end{cases} \tag{4}$$

$\sqsubseteq^{\mathrm{bw}}$ is an efficiently computable GFQ preorder [32] incomparable with forward simulations. It can be used to establish language inclusion by matching final states of $\mathcal{A}$ with final states of $\mathcal{B}$ (dually to forward simulations); in this sense, it is GFI.

Lemma 2.3 ([32])**.** *Backward sim. is a PTIME GFQ/GFI preorder.*

The corresponding notion of *backward trace inclusion* $\subseteq^{\mathrm{bw}}$ is defined as follows: $p \subseteq^{\mathrm{bw}} q$ iff, for every finite word $w = \sigma_0\sigma_1 \cdots \sigma_{m-1} \in \Sigma^*$, and for every initial, finite w-trace $\pi_0 = p_0 \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{m-1}} p_m$ ending in $p_m = p$, there exists an initial, finite w-trace $\pi_1 = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{m-1}} q_m$ ending in $q_m = q$, s.t., for any $i \geq 0$, if $p_i \in F$, then $q_i \in F$. Note that backward trace inclusion deals with *finite traces* (unlike forward trace inclusions), which is due to the asymmetry between past and future in ω-automata. Clearly, $\sqsubseteq^{\mathrm{bw}} \subseteq \subseteq^{\mathrm{bw}}$; we observe that even $\subseteq^{\mathrm{bw}}$ is GFQ/GFI.

Theorem 2.4. *Backward trace inclusion is a GFQ/GFI preorder.*

Proof. We first prove that $\subseteq^{\mathrm{bw}}$ is GFQ. Let $\sqsubseteq := \subseteq^{\mathrm{bw}}$. Let $w = \sigma_0\sigma_1 \cdots \in \mathcal{L}(\mathcal{A}/\sqsubseteq)$, and we show $w \in \mathcal{L}(\mathcal{A})$. There exists an initial, infinite and fair w-trace $\pi = [q_0] \xrightarrow{\sigma_0} [q_1] \xrightarrow{\sigma_1} \cdots$. For $i \geq 0$, let $w_i = \sigma_0\sigma_1 \cdots \sigma_i$ (with $w_{-1} = \varepsilon$), and let $\pi[0..i]$ be the w_{i-1}-trace

[3] Delayed fixed-word simulation is defined as a variant of simulation where Duplicator has ω-lookahead only on the input word w, and *not* on Spoiler's actual w-trace π_0; that it subsumes $\subseteq^{\mathrm{di}}$ is non-trivial.

prefix of π. For any $i \geq 0$, we build by induction an initial, finite w_{i-1}-trace π_i ending in q_i (of length i) visiting at least as many accepting states as $\pi[0..i]$ (and at the same time $\pi[0..i]$ does).

For $i = 0$, just take the empty ε-trace $\pi_0 = q_0$. For $i > 0$, assume that an initial w_{i-2}-trace π_{i-1} ending in q_{i-1} has already been built. We have the transition $[q_{i-1}] \xrightarrow{\sigma_{i-1}} [q_i]$ in $\mathcal{L}(\mathcal{A}/\sqsubseteq)$. There exist $\hat{q} \in [q_{i-1}]$ and $\hat{q}' \in [q_i]$ s.t. we have a transition $\hat{q} \xrightarrow{\sigma_{i-1}} \hat{q}'$ in $\mathcal{A}$. W.l.o.g. we can assume that $\hat{q}' = q_i$, since $[q_i] = [\hat{q}']$. By $q_{i-1} \subseteq^{\mathsf{bw}} \hat{q}$, there exists an initial, finite w_{i-2}-trace π' ending in $\hat{q}$. By the definition of backward inclusion, π' visits at least as many accepting states as π_{i-1}, which, by inductive hypothesis, visits at least as many accepting states as $\pi[0..i-1]$. Therefore, $\pi_i := \pi' \xrightarrow{\sigma_{i-1}} q_i$ is an initial, finite w_{i-1}-trace ending in q_i. Moreover, if $[q_i] \in F'$, then, since backward inclusion respects accepting states, $[q_i] \subseteq F$, hence $q_i \in F$, and, consequently, π_i visits at least as many accepting states as $\pi[0..i]$. Since π is fair, the finite, initial traces $\pi_0, \pi_1, \cdots$ visit unboundedly many accepting states. Since $\mathcal{A}$ is finitely branching, by König's Lemma there exists an initial, infinite and fair w-trace π_ω. Therefore, $w \in \mathcal{L}(\mathcal{A})$.

We now prove that $\subseteq^{\mathsf{bw}}$ is GFI. Let $\mathcal{A}$ and $\mathcal{B}$ be two automata. For backward notions, we require that every accepting state in $\mathcal{A}$ is in relation with an accepting state in $\mathcal{B}$. Let $w = \sigma_0\sigma_1 \cdots \in \mathcal{L}(\mathcal{A})$, and let $\pi_0 = p_0 \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_0} \cdots$ be an initial and fair w-path in $\mathcal{A}$. Since π_0 visits infinitely many accepting states, and since each such state is $\subseteq^{\mathsf{bw}}$-related to an accepting state in $\mathcal{B}$, by using the definition of $\subseteq^{\mathsf{bw}}$ it is possible to build in $\mathcal{B}$ longer and longer finite initial traces in $\mathcal{B}$ visiting unboundedly many accepting states. Since $\mathcal{B}$ is finitely branching, by König's Lemma there exists an infinite, initial and fair w-trace π_ω in $\mathcal{B}$. Thus, $w \in \mathcal{L}(\mathcal{B})$. □

3. Transition Pruning Minimization Techniques

While quotienting-based minimization techniques reduce the number of states by merging them, we explore an alternative method which prunes (i.e., removes) transitions. The intuition is that certain transitions can be removed from an automaton without changing its language when other 'better' transitions remain.

Definition Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a BA and let P a transitive, asymmetric relation on δ. The *pruned automaton* is defined as $Prune(\mathcal{A}, P) := (\Sigma, Q, I, F, \delta')$, with $\delta' = \{(p,\sigma,r) \in \delta \mid \nexists (p',\sigma',r') \in \delta \cdot (p,\sigma,r) P (p',\sigma',r')\}$.

By the assumptions on P, the pruned automaton $Prune(\mathcal{A}, P)$ is uniquely defined. Notice that transitions are removed 'in parallel'. Though P might depend on δ, P is *not* re-computed even if the removal of a single transition changes δ. This is important because computing P may be expensive. Since removing transitions cannot introduce new words in the language, $Prune(\mathcal{A}, P) \subseteq \mathcal{A}$. When also the converse inclusion holds (so the language is preserved), we say that P is *good for pruning* (GFP), i.e., P is GFP iff $Prune(\mathcal{A}, P) \approx \mathcal{A}$. Clearly, GFP is $\subseteq$-downward closed (like GFI and GFQ).

We study GFP relations obtained by comparing the endpoints of transitions over the same input symbol. Formally, given two binary relations $R_b, R_f \subseteq Q \times Q$, we define

$$P(R_b, R_f) = \{((p,\sigma,r),(p',\sigma,r')) \mid pR_bp' \text{ and } rR_fr'\}$$

$P(\cdot,\cdot)$ is monotone in both arguments. In the following, we explore which state relations R_b, R_f induce GFP relations $P(R_b, R_f)$.

It has long been known that $P(id, \sqsubset^{\mathsf{di}})$ and $P(\sqsubset^{\mathsf{bw}}, id)$ are GFP (see [7] where the removed transitions are called 'little brothers'). Moreover, even the relation $R_t(\subset^{\mathsf{f}}) := P(id, \sqsubset^{\mathsf{di}}) \cup \{((p,\sigma,r),(p,\sigma,r')) \mid (p,\sigma,r') \text{ is transient and } r \subset^{\mathsf{f}} r'\}$ is GFP [32], i.e., strict fair trace inclusion suffices if the remaining transition can only be used once. However, in general, $P(id, \subset^{\mathsf{f}})$ is not GFP. Moreover, even if only transient transitions are compared/pruned, $P(\sqsubset^{\mathsf{bw}}, \subset^{\mathsf{f}})$ is not GFP; cf. Fig. 2.

$R_b \backslash R_f$	id	$\sqsubset^{\mathsf{di}}$	$\subset^{\mathsf{di}}$	$\sqsubset^{\mathsf{de}}$	$\sqsubset^{\mathsf{f}}$
id	×	✓	✓	×	×
$\sqsubset^{\mathsf{bw}}$	✓	✓	✓	×	×
$\subset^{\mathsf{bw}}$	✓	✓	×	×	×

Figure 1. GFP relations $P(R_b, R_f)$

Theorem 3.1. *For every asymmetric and transitive relation $R \subseteq \subseteq^{\mathsf{di}}$, $P(id, R)$ is GFP.*

Proof. Let $\mathcal{A}' = Prune(\mathcal{A}, P(id, R))$. We show $\mathcal{A} \subseteq \mathcal{A}'$. If $w = \sigma_0\sigma_1 \cdots \in \mathcal{L}(\mathcal{A})$ then there exists an infinite fair initial trace $\hat{\pi}$ on w in $\mathcal{A}$. We show $w \in \mathcal{L}(\mathcal{A}')$.

We call a trace $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ on w in $\mathcal{A}$ *i-good* if it does not contain any transition $q_j \xrightarrow{\sigma_j} q_{j+1}$ for $j < i$ s.t. there exists an $\mathcal{A}$ transition $q_j \xrightarrow{\sigma_j} q'_{j+1}$ with $q_{j+1} R q'_{j+1}$ (i.e., no such transition is used within the first i steps). Since $\mathcal{A}$ is finitely branching, for every state and symbol there exists at least one R-maximal successor that is still present in $\mathcal{A}'$, because R is asymmetric and transitive. Thus, for every i-good trace π on w there exists an $(i+1)$-good trace π' on w s.t. π and π' are identical on the first i steps and $\mathcal{C}^{\mathsf{di}}(\pi, \pi')$, because $R \subseteq \subseteq^{\mathsf{di}}$. Since $\hat{\pi}$ is an infinite fair initial trace on w (which is trivially 0-good), there exists an infinite initial trace $\tilde{\pi}$ on w that is i-good for every i and $\mathcal{C}^{\mathsf{di}}(\hat{\pi}, \tilde{\pi})$. Moreover, $\tilde{\pi}$ is a trace in $\mathcal{A}'$. Since $\hat{\pi}$ is fair and $\mathcal{C}^{\mathsf{di}}(\hat{\pi}, \tilde{\pi})$, $\tilde{\pi}$ is an infinite fair initial trace on w that is i-good for every i. Therefore $\tilde{\pi}$ is a fair initial trace on w in $\mathcal{A}'$ and thus $w \in \mathcal{L}(\mathcal{A}')$. □

Theorem 3.2. *For every asymmetric and transitive relation $R \subseteq \subseteq^{\mathsf{bw}}$, $P(R, id)$ is GFP.*

Proof. Let $\mathcal{A}' = Prune(\mathcal{A}, P(R, id))$. We show $\mathcal{A} \subseteq \mathcal{A}'$. If $w = \sigma_0\sigma_1 \cdots \in \mathcal{L}(\mathcal{A})$ then there exists an infinite fair initial trace $\hat{\pi}$ on w in $\mathcal{A}$. We show $w \in \mathcal{L}(\mathcal{A}')$.

We call a trace $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ on w in $\mathcal{A}$ *i-good* if it does not contain any transition $q_j \xrightarrow{\sigma_j} q_{j+1}$ for $j < i$ s.t. there exists an $\mathcal{A}$ transition $q'_j \xrightarrow{\sigma_j} q_{j+1}$ with $q_j R q'_j$ (i.e., no such transition is used within the first i steps).

We show, by induction on i, the following property (P): For every i and every initial trace π on w in $\mathcal{A}$ there exists an initial i-good trace π' on w in $\mathcal{A}$ s.t. π and π' have identical suffixes from step i onwards and $\mathcal{C}^{\mathsf{di}}(\pi, \pi')$.

The base case $i = 0$ is trivial with $\pi' = \pi$. For the induction step there are two cases. If π is $(i+1)$-good then we can take $\pi' = \pi$. Otherwise there exists a transition $q'_i \xrightarrow{\sigma_i} q_{i+1}$ with $q_i R q'_i$. Without restriction (since $\mathcal{A}$ is finite and R is asymmetric and transitive) we assume that q'_i is R-maximal among the σ_i-predecessors of q_{i+1}. In particular, the transition $q'_i \xrightarrow{\sigma_i} q_{i+1}$ is present in $\mathcal{A}'$. Since $R \subseteq \subseteq^{\mathsf{bw}}$, there exists an initial trace π'' on w that has suffix $q'_i \xrightarrow{\sigma_i} q_{i+1} \xrightarrow{\sigma_{i+1}} q_{i+2} \ldots$ and $\mathcal{C}^{\mathsf{di}}(\pi, \pi'')$. Then, by induction hypothesis, there exists an initial i-good trace π' on w that has suffix $q'_i \xrightarrow{\sigma_i} q_{i+1} \xrightarrow{\sigma_{i+1}} q_{i+2} \ldots$ and $\mathcal{C}^{\mathsf{di}}(\pi'', \pi')$. Since q'_i is R-maximal among the σ_i-predecessors of q_{i+1} we obtain that π' is also $(i+1)$-good. Moreover, π' and π have identical suffixes from step $i+1$ onwards. Finally, by $\mathcal{C}^{\mathsf{di}}(\pi, \pi'')$ and $\mathcal{C}^{\mathsf{di}}(\pi'', \pi')$, we obtain $\mathcal{C}^{\mathsf{di}}(\pi, \pi')$.

Given the infinite fair initial trace $\hat{\pi}$ on w in $\mathcal{A}$, it follows from property (P) and König's Lemma that there exists an infinite initial

trace $\tilde{\pi}$ on w that is i-good for every i and $\mathcal{C}^{\mathsf{di}}(\hat{\pi},\tilde{\pi})$. Therefore $\tilde{\pi}$ is an infinite fair initial trace on w in $\mathcal{A}'$ and thus $w \in \mathcal{L}(\mathcal{A}')$. □

Theorem 3.3. *If $\mathcal{A} = \mathcal{A}/\sqsubseteq^{\mathsf{bw}}$ then $P(\sqsubset^{\mathsf{bw}}, \subseteq^{\mathsf{di}})$ is GFP.*

Proof. Let $\mathcal{A}' = \mathit{Prune}(\mathcal{A}, P(\sqsubset^{\mathsf{bw}}, \subseteq^{\mathsf{di}}))$. We show $\mathcal{A} \subseteq \mathcal{A}'$. Let $w = \sigma_0\sigma_1\cdots \in \mathcal{L}(\mathcal{A})$. Then there exists an infinite fair initial trace $\hat{\pi}$ on w in $\mathcal{A}$. We show $w \in \mathcal{L}(\mathcal{A}')$.

We call a trace $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ on w start-maximal iff it is initial and there does not exist any trace $\pi' = q'_0 \xrightarrow{\sigma_0} q'_1 \xrightarrow{\sigma_1} \cdots$ on w s.t. $\mathcal{C}^{\mathsf{di}}(\pi,\pi')$ and $q_0 \sqsubset^{\mathsf{bw}} q'_0$. We call a trace $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ on w *i-good* iff it is start-maximal and π does not contain any transition $q_j \xrightarrow{\sigma_j} q_{j+1}$ for $j < i$ s.t. there exists an $\mathcal{A}$ transition $q_j \xrightarrow{\sigma_j} q'_{j+1}$ with $q_{j+1} \sqsubset^{\mathsf{bw}} q'_{j+1}$ and there exists an infinite trace $\pi'[j+1..]$ from q'_{j+1} with $\mathcal{C}^{\mathsf{di}}(\pi[j+1..],\pi'[j+1..])$.

Since $\mathcal{A}$ is finite, there are $\sqsubset^{\mathsf{bw}}$-maximal elements among those finitely many successors of every state q_j from which there exists an infinite trace $\pi'[j+1..]$ with $\mathcal{C}^{\mathsf{di}}(\pi[j+1..],\pi'[j+1..])$. Thus, for every infinite i-good trace π on w there exists an $(i+1)$-good trace π' on w s.t. π and π' are identical on the first i steps and $\mathcal{C}^{\mathsf{di}}(\pi,\pi')$.

Since there is an infinite fair initial trace $\hat{\pi}$ on w, there also exists a start-maximal, and thus 0-good, fair initial trace on w, because $\sqsubset^{\mathsf{bw}}$ has maximal elements. Then it follows from the property above that there exists an infinite initial trace $\tilde{\pi}$ on w that is i-good for every i and $\mathcal{C}^{\mathsf{di}}(\hat{\pi},\tilde{\pi})$. In particular, this implies that $\tilde{\pi}$ is fair. So $\tilde{\pi}$ is an infinite fair initial trace on w that is i-good for every i.

Let now $\tilde{\pi} = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$. We show that $\tilde{\pi}$ is also possible in $\mathcal{A}'$ by assuming the opposite and deriving a contradiction. Suppose that $\tilde{\pi}$ contains a transition $q_j \xrightarrow{\sigma_j} q_{j+1}$ that is not present in $\mathcal{A}'$. Then there must exist a transition $q'_j \xrightarrow{\sigma_j} q'_{j+1}$ in $\mathcal{A}'$ s.t. $q_j \sqsubset^{\mathsf{bw}} q'_j$ and $q_{j+1} \subseteq^{\mathsf{di}} q'_{j+1}$. We cannot have $j = 0$, because in this case $\tilde{\pi}$ would not be start-maximal and thus not even 1-good. So we get $j \geq 1$. Since $q_j \sqsubset^{\mathsf{bw}} q'_j$ and $q_{j-1} \xrightarrow{\sigma_{j-1}} q_j$ there must exist a state q'_{j-1} s.t. $q'_{j-1} \xrightarrow{\sigma_{j-1}} q'_j$ and $q_{j-1} \sqsubseteq^{\mathsf{bw}} q'_{j-1}$. In particular, $q_x \in F \Rightarrow q'_{x+1} \in F$ for $x \in \{j-1, j\}$. By $\mathcal{A} = \mathcal{A}/\sqsubseteq^{\mathsf{bw}}$ we obtain that either $q_{j-1} = q'_{j-1}$ or $q_{j-1} \sqsubset^{\mathsf{bw}} q'_{j-1}$. The first case would imply that π' is not j-good, because $q_{j+1} \subseteq^{\mathsf{di}} q'_{j+1}$, and thus yield a contradiction. Therefore, we must have $q_{j-1} \sqsubset^{\mathsf{bw}} q'_{j-1}$. We cannot have $j-1 = 0$, because in this case π' would not be start-maximal and thus not even 1-good. So we get $j-1 \geq 1$. The whole argument above repeats with $j-1, j-2, j-3, \ldots$ substituted for j until we get a contradiction or 0 is reached. Reaching 0 also yields a contradiction to start-maximality of $\tilde{\pi}$, as above. Therefore $\tilde{\pi}$ is a fair initial trace on w in $\mathcal{A}'$ and thus $w \in \mathcal{L}(\mathcal{A}')$. □

Theorem 3.4. *$P(\subseteq^{\mathsf{bw}}, \sqsubset^{\mathsf{di}})$ is GFP.*

Proof. Let $\mathcal{A}' = \mathit{Prune}(\mathcal{A}, P(\subseteq^{\mathsf{bw}}, \sqsubset^{\mathsf{di}}))$. We show $\mathcal{A} \subseteq \mathcal{A}'$. Let $w = \sigma_0\sigma_1\cdots \in \mathcal{L}(\mathcal{A})$. Then there exists an infinite fair initial trace $\hat{\pi}$ on w in $\mathcal{A}$. We show $w \in \mathcal{L}(\mathcal{A}')$.

Given some infinite initial trace $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ on w, we call it i-good iff its first i transitions are also possible in $\mathcal{A}'$.

We now show, by induction on i, the following property (P): For every infinite initial trace $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ on w and every $i \geq 0$, there exists an infinite initial trace $\pi' = q'_0 \xrightarrow{\sigma_0} q'_1 \xrightarrow{\sigma_1} \cdots$ on w that is i-good and $\mathcal{C}^{\mathsf{di}}(\pi,\pi')$ and $\forall j \geq i.\, q_j \sqsubseteq^{\mathsf{di}} q'_j$.

The base case $i = 0$ is trivially true with $\pi' = \pi$. For the induction step consider an infinite initial trace $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ on w. By induction hypothesis, there exists an infinite initial trace $\pi^1 = q^1_0 \xrightarrow{\sigma_0} q^1_1 \xrightarrow{\sigma_1} \cdots$ on w that is i-good and $\mathcal{C}^{\mathsf{di}}(\pi,\pi^1)$ and $\forall j \geq i.\, q_j \sqsubseteq^{\mathsf{di}} q^1_j$.

If π^1 is $(i+1)$-good then we are done. Otherwise, the transition $q^1_i \xrightarrow{\sigma_i} q^1_{i+1}$ is not present in $\mathcal{A}'$. Since $\mathcal{A}' = \mathit{Prune}(\mathcal{A}, P(\subseteq^{\mathsf{bw}}, \sqsubset^{\mathsf{di}}))$, there must exist a transition $q^2_i \xrightarrow{\sigma_i} q^2_{i+1}$ in $\mathcal{A}'$ s.t. $q^1_i \subseteq^{\mathsf{bw}} q^2_i$ and $q^1_{i+1} \sqsubset^{\mathsf{di}} q^2_{i+1}$. It follows from the definitions of $\subseteq^{\mathsf{bw}}$ and $\sqsubset^{\mathsf{di}}$ that there exists an infinite initial trace $\pi^2 = q^2_0 \xrightarrow{\sigma_0} q^2_1 \xrightarrow{\sigma_1} \cdots$ on w s.t. $\mathcal{C}^{\mathsf{di}}(\pi^1,\pi^2)$, $q^1_{i+1} \sqsubset^{\mathsf{di}} q^2_{i+1}$ and $\forall j \geq i+1.\, q^1_j \sqsubseteq^{\mathsf{di}} q^2_j$. (This last property uses the fact that $\sqsubseteq^{\mathsf{di}}$ propagates forward. Direct trace inclusion $\subseteq^{\mathsf{di}}$ does not suffice.) By induction hypothesis, there exists an infinite initial trace $\pi^3 = q^3_0 \xrightarrow{\sigma_0} q^3_1 \xrightarrow{\sigma_1} \cdots$ on w that is i-good and $\mathcal{C}^{\mathsf{di}}(\pi^2,\pi^3)$ and $\forall j \geq i.\, q^2_j \sqsubseteq^{\mathsf{di}} q^3_j$. By transitivity we obtain $\mathcal{C}^{\mathsf{di}}(\pi^1,\pi^3)$, $q^1_{i+1} \sqsubset^{\mathsf{di}} q^3_{i+1}$ and $\forall j \geq i+1.\, q^1_j \sqsubseteq^{\mathsf{di}} q^3_j$.

If π^3 is $(i+1)$-good then we are done. Otherwise, the argument of the above paragraph repeats and we obtain an infinite initial trace $\pi^5 = q^5_0 \xrightarrow{\sigma_0} q^5_1 \xrightarrow{\sigma_1} \cdots$ on w that is i-good and $\mathcal{C}^{\mathsf{di}}(\pi^3,\pi^5)$, that $q^3_{i+1} \sqsubset^{\mathsf{di}} q^5_{i+1}$ and $\forall j \geq i+1.\, q^3_j \sqsubseteq^{\mathsf{di}} q^5_j$. This process cannot repeat infinitely often, because this would imply an infinite strictly increasing $\sqsubset^{\mathsf{di}}$-chain q^{2x+1}_{i+1} for $x = 0, 1, 2, \ldots$, which is impossible in finite automata. Therefore, for some finite index x, we obtain an infinite initial trace $\pi^x = q^x_0 \xrightarrow{\sigma_0} q^x_1 \xrightarrow{\sigma_1} \cdots$ on w that is $(i+1)$-good and, by transitivity, $\mathcal{C}^{\mathsf{di}}(\pi,\pi^x)$ and $\forall j \geq i+1.\, q_j \sqsubseteq^{\mathsf{di}} q^x_j$. Thus π^x is the trace π' that we were looking for.

Given the infinite fair initial trace $\hat{\pi}$ on w in $\mathcal{A}$, it follows from property (P) and König's Lemma that there exists an infinite initial trace $\tilde{\pi}$ on w that is i-good for every i and $\mathcal{C}^{\mathsf{di}}(\hat{\pi},\tilde{\pi})$. Therefore $\tilde{\pi}$ is an infinite fair initial trace on w in $\mathcal{A}'$ and thus $w \in \mathcal{L}(\mathcal{A}')$. □

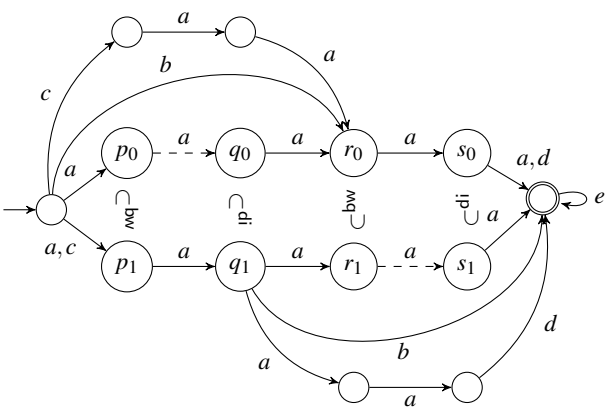
Theorem 3.4 implies that $P(\sqsubset^{\mathsf{bw}}, \sqsubset^{\mathsf{di}})$ is GFP, but $P(\subset^{\mathsf{bw}}, \subset^{\mathsf{di}})$ is not; see Figure 2. Moreover, $P(\mathit{id}, \sqsubset^{\mathsf{de}})$ is not GFP (even if $\mathcal{A} = \mathcal{A}/\sqsubseteq^{\mathsf{de}}$); see Figure 2.

The quotienting and transition pruning techniques described above use intricate combinations of backward and forward simulations (and more general trace inclusions). In particular, they subsume previous attempts to combine backward and forward simulations for automata minimization by *mediated preorder* [5] (but not vice-versa). Mediated preorder is defined as the largest fragment $M \subseteq \sqsubseteq^{\mathsf{di}} \circ (\sqsubseteq^{\mathsf{bw}})^{-1}$ s.t. $M \circ \sqsubseteq^{\mathsf{di}} \subseteq M$. In particular, M is a preorder that is GFQ. However, an automaton $\mathcal{A}$ that has been minimized by the techniques described above cannot be further reduced by mediated preorder. First we have $\mathcal{A} = \mathcal{A}/\sqsubseteq^{\mathsf{bw}} = \mathcal{A}/\sqsubseteq^{\mathsf{di}}$ by repeated quotienting. Second, there cannot exist any distinct states x, y in $\mathcal{A}$ s.t. $(x \sqsubseteq^{\mathsf{di}} y \wedge x \sqsubseteq^{\mathsf{bw}} y)$ by the pruning techniques above (used with simulations as approximations for trace inclusions) and the removal of dead states. Under these conditions, quotienting with mediated preorder has no effect, as the following theorem shows.

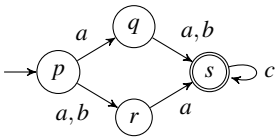
Theorem 3.5. *Let $\mathcal{A}$ be an automaton s.t. (1) $\mathcal{A} = \mathcal{A}/\sqsubseteq^{\mathsf{bw}} = \mathcal{A}/\sqsubseteq^{\mathsf{di}}$ and (2) $x \sqsubseteq^{\mathsf{di}} y \wedge x \sqsubseteq^{\mathsf{bw}} y \Rightarrow x = y$. Then $\mathcal{A} = \mathcal{A}/M$.*

Proof. We show that $xMy \wedge yMx \Rightarrow x = y$ which implies $\mathcal{A} = \mathcal{A}/M$.

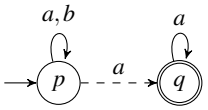
Let xMy and yMx. By definition of M there exist mediators z s.t. $x \sqsubseteq^{\mathsf{di}} z$ and $y \sqsubseteq^{\mathsf{bw}} z$, and w s.t. $x \sqsubseteq^{\mathsf{bw}} w$ and $y \sqsubseteq^{\mathsf{di}} w$. Since $M \circ \sqsubseteq^{\mathsf{di}} \subseteq M$ we have xMw. Thus there exists a mediator k s.t. $x \sqsubseteq^{\mathsf{di}} k$ and $w \sqsubseteq^{\mathsf{bw}} k$. By transitivity of $\sqsubseteq^{\mathsf{bw}}$ we also have $x \sqsubseteq^{\mathsf{bw}} k$. By (2) we get $x = k$. Thus $x \sqsubseteq^{\mathsf{bw}} w$ and $w \sqsubseteq^{\mathsf{bw}} x$. By (1) we get $x = w$. Thus $y \sqsubseteq^{\mathsf{di}} w = x \sqsubseteq^{\mathsf{di}} z$ and by transitivity $y \sqsubseteq^{\mathsf{di}} z$. Moreover, $y \sqsubseteq^{\mathsf{bw}} z$ as above. By (2) we get $z = y$. Thus $x \sqsubseteq^{\mathsf{di}} z = y$ and $y \sqsubseteq^{\mathsf{di}} w = x$. By (1) we get $x = y$. □



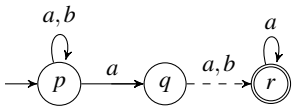
(a) $P(\subset^{\mathsf{bw}}, \subset^{\mathsf{di}})$ is not GFP: If the dashed transitions $p_0 \xrightarrow{a} q_0$ and $r_1 \xrightarrow{a} s_1$ are removed, then a^5e^ω is no longer accepted. Note that $\mathcal{A} = \mathcal{A}/\subseteq^{\mathsf{bw}} = \mathcal{A}/\subseteq^{\mathsf{di}}$. (This example even holds for $\prec^{k\text{-}\mathsf{bw}}$, $\prec^{k\text{-}\mathsf{di}}$ and $k = 3$; cf. Section 4).



(b) GFP is not closed under union: Pruning automaton $\mathcal{A}$ with $P(id, \sqsubset^{\mathsf{di}}) \cup P(\sqsubset^{\mathsf{bw}}, id)$ would remove the transitions $p \xrightarrow{a} r$ and $q \xrightarrow{a} s$, and thus aac^ω would no longer be accepted.



(c) $P(id, \sqsubset^{\mathsf{de}})$ is not GFP: We have $q \sqsubset^{\mathsf{de}} p$, but removing the dashed transition $p \xrightarrow{a} q$ makes the language empty, even though $\mathcal{A} = \mathcal{A}/\sqsubseteq^{\mathsf{de}}$.



(d) $P(\sqsubset^{\mathsf{bw}}, \subset^{\mathsf{f}})$ is not GFP: In the automaton above, both transitions $p \xrightarrow{a} q$ and $q \xrightarrow{a} r$ are transient. Moreover, $r \subset^{\mathsf{f}} q$ (even $r \sqsubset^{\mathsf{de}} q$) and $q \sqsubset^{\mathsf{bw}} p$. However, removing the smaller transition $q \xrightarrow{a} r$ changes the language, since a^ω is no longer accepted. Thus, $P(\sqsubset^{\mathsf{bw}}, \subset^{\mathsf{f}})$ is not GFP even when one restricts to comparing/pruning only transient transitions (unlike $P(id, \subset^{\mathsf{f}})$).

Figure 2. Pruning counterexamples.

4. Lookahead Simulations

While trace inclusions are theoretically appealing as GFQ/GFI preorders coarser than simulations, it is not feasible to use them in practice, because they are too hard to compute (even their membership problem is PSPACE-complete). As a first attempt at achieving a better trade-off between complexity and size we recall *multipebble simulations* [12], which are obtained by providing Duplicator with several pebbles, instead of one. However, computing multipebble simulations is not feasible in practice either, on automata of nontrivial size. Therefore, we explore yet another way of obtaining good under-approximations of trace inclusion: We introduce *lookahead simulations*, which are obtained by providing Duplicator with a limited amount of information about Spoiler's future moves. While lookahead itself is a classic concept (e.g., in parsing) it can be defined in several different ways in the context of adversarial games like in simulation. We compare different variants for computational efficiency and approximation quality.

***k*-pebble simulation.** Simulation preorder can be generalized by allowing Duplicator to control several pebbles instead of just one. In k-pebble simulation, $k > 0$, Duplicator's position is a set of at most k states (while Spoiler still controls exactly 1 state), which allows Duplicator to 'hedge her bets' in the simulation game. The direct, delayed, fair and backward winning conditions can be generalized to the multipebble framework [12]. For $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$ and $k > 0$, k-pebble x-simulation is coarser than x-simulation and it implies x-containment; by increasing k, one can control the quality of the approximation to trace inclusion. Direct, delayed, fair and backward k-pebble simulations are not transitive in general, but their transitive closures are GFI preorders; the direct, delayed and backward variants are also GFQ. However, computing k-pebble simulations is infeasible, even for modest values for k. In fact, for a BA with n states, computing k-pebble simulation requires solving a game of size $n \cdot n^k$. Even in the simplest case of $k = 2$ this means at least cubic space, which is not practical for large n. For this reason, we consider a different way to extend Duplicator's power, i.e., by using *lookahead* on the moves of Spoiler.

***k*-step simulation.** We generalize simulation by having the players select sequences of transitions of length $k > 0$ instead of single transitions: This gives Duplicator more information, and thus yields a larger simulation relation. In general, k-step simulation and k-pebble simulation are incomparable, but k-step simulation is strictly contained in n-pebble simulation. However, the rigid use of lookahead in big-steps causes at least two issues: 1) For a BA with n states, we need to store only n^2 configurations (p,q) (which is much less than k-pebble simulation). However, in *every round* we have to explore up-to d^k different moves for each player (where d is the maximal out-degree of the automaton). In practice (e.g., $d = 4$, $k = 12$) this is still too large. 2) Duplicator's lookahead varies between 1 and k, depending where she is in her response to Spoiler's long move. Thus, Duplicator might lack lookahead where it is most needed, while having a large lookahead in other situations where it is not useful. In the next notion, we attempt at ameliorating this.

***k*-continuous simulation.** Duplicator is continuously kept informed about Spoiler's next k moves, i.e., she always has lookahead k. Formally, a configuration of the simulation game consists in a pair (ρ_i, q_i), where ρ_i is the sequence of the next $k-1$ moves from p_i that Spoiler has already committed to. In every round of the game, Spoiler reveals another move k steps in the future, and then makes the first of her announced k moves, to which Duplicator responds as usual. A pair of states (p,q) is in k-continuous simulation if Duplicator can win this game from every configuration (ρ, q), where ρ is a sequence of $k-1$ moves from p. ($k = 1$ is ordinary simulation.) k-continuous simulation is strictly contained in n-pebble simulation (but incomparable with k-pebble simulation), and larger than k-step simulation. While this is arguably the strongest way of giving lookahead to Duplicator, it requires storing $n^2 \cdot d^{k-1}$ configurations, which is infeasible for nontrivial n and k (e.g., $n = 10000$, $d = 4$, $k = 12$).

***k*-lookahead simulation.** We introduce k-lookahead simulation as an optimal compromise between k-step and k-continuous simulation. Intuitively, we put the lookahead under Duplicator's control, who can choose at each round how much lookahead she needs (up to k). Formally, configurations are pairs (p_i, q_i) of states. In every round of the game, Spoiler chooses a sequence of k consecutive transitions $p_i \xrightarrow{\sigma_i} p_{i+1} \xrightarrow{\sigma_{i+1}} \cdots \xrightarrow{\sigma_{i+k-1}} p_{i+k}$. Duplicator then chooses a number $1 \leq m \leq k$ and responds with a matching sequence of m transitions $q_i \xrightarrow{\sigma_i} q_{i+1} \xrightarrow{\sigma_{i+1}} \cdots \xrightarrow{\sigma_{i+m-1}} q_{i+m}$. The remaining $k-m$ moves of Spoiler are forgotten, and the next round of the game starts at (p_{i+m}, q_{i+m}). In this way, the players build two infinite traces π_0 from p_0 and π_1 from q_0. Backward simulation is de-

fined similarly with backward transitions. For acceptance condition $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$, Duplicator wins this play if $\mathcal{C}^x(\pi_0, \pi_1)$ holds.

Definition Two states (p_0, q_0) are in *k-lookahead x-simulation*, written $p_0 \sqsubseteq^{k\text{-}x} q_0$, iff Duplicator has a winning strategy in the above game.

Since $\sqsubseteq^{k\text{-}x}$ is not transitive (unless $k = 1$; cf. Appendix B), we denote its transitive closure, which is a preorder, by $\preceq^{k\text{-}x}$, and its asymmetric restriction by $\prec^{k\text{-}x} = \preceq^{k\text{-}x} \setminus (\succeq^{k\text{-}x})^{-1}$.

Lookahead simulation offers the optimal trade-off between k-step and k-continuous simulation. Since the lookahead is discarded at each round, k-lookahead simulation is (strictly) included in k-continuous lookahead (where the lookahead is never discarded). However, this has the benefit of only requiring to store n^2 configurations, which makes computing lookahead simulation space-efficient. On the other side, when Duplicator always chooses a maximal reply $m = k$ we recover k-step simulation, which is thus included in k-lookahead simulation. Moreover, thanks to the fact that Duplicator controls the lookahead, most rounds of the game can be solved without ever reaching the maximal lookahead k: 1) for a fixed attack by Spoiler, we only consider Duplicator's responses for small $m = 1, 2, \ldots, k$ until we find a winning one, and 2) also Spoiler's attacks can be built incrementally since, if she loses for some lookahead h, then she also loses for $h' > h$. In practice, this greatly speeds up the computation, and allows us to use lookaheads in the range 4-25, depending on the size and structure of the automata; see Section 7 for the experimental evaluation and benchmark against the GOAL tool [34].

k-lookahead simulation can also be expressed as a restriction of n-pebble simulation, where Duplicator is allowed to split pebbles maximally (thus n-pebbles), but after a number $m \leq k$ rounds (where m is chosen dynamically by Duplicator) she has to discard all but one pebble. Then, Duplicator is allowed to split pebbles maximally again, etc. Thus, k-lookahead simulation is contained in n-pebble simulation, though it is generally incomparable with k-pebble simulation.

Direct, delayed, fair and backward k-lookahead simulation have a fixed-point characterization expressible in μ-calculus (cf. Appendix C), which can be useful for a symbolic implementation. However, our current algorithm computes them with an explicit-state representation.

5. Automata Minimization

We minimize automata by transition pruning and quotienting. While trace inclusions would be an ideal basis for such techniques, they (i.e., their membership problems) are PSPACE-complete. Instead, we use lookahead simulations as efficiently computable under-approximations; in particular, we use

- $\preceq^{k\text{-}\mathsf{di}}$ in place of direct trace inclusion $\subseteq^{\mathsf{di}}$ (which is GFQ [12]).
- $\preceq^{k\text{-}\mathsf{de}}$ in place of n-pebble delayed simulation (GFQ [12]).
- $\preceq^{k\text{-}\mathsf{f}}$ in place of fair trace inclusion $\subseteq^{\mathsf{f}}$ (which is GFI).
- $\preceq^{k\text{-}\mathsf{bw}}$ in place of backward trace inclusion $\subseteq^{\mathsf{bw}}$ (which is GFQ by Theorem 2.4).

For pruning, we apply the results of Section 3 and the substitutions above to obtain the following GFP relations:

$$P(id, \prec^{k\text{-}\mathsf{di}}), P(\prec^{k\text{-}\mathsf{bw}}, id), P(\sqsubset^{\mathsf{bw}}, \preceq^{k\text{-}\mathsf{di}}), P(\preceq^{k\text{-}\mathsf{bw}}, \sqsubset^{\mathsf{di}}), R_t(\prec^{k\text{-}\mathsf{f}})$$

For quotienting, we employ delayed $\preceq^{k\text{-}\mathsf{de}}$ and backward $\preceq^{k\text{-}\mathsf{bw}}$ k-lookahead simulations (which are GFQ). Below, we describe two possible ways to combine our simplification techniques: *Heavy-k* and *Light-k* (which are parameterized by the lookahead value k).

Heavy-k. We advocate the following minimization procedure, which repeatedly applies all the techniques described in this paper until a fixpoint is reached: 1) Remove dead states. 2) Prune transitions w.r.t. the GFP relations above (using lookahead k). 3) Quotient w.r.t. $\preceq^{k\text{-}\mathsf{de}}$ and $\preceq^{k\text{-}\mathsf{bw}}$. The resulting simplified automaton cannot be further reduced by any of these techniques. In this sense, it is a local minimum in the space of automata. Applying the techniques in a different order might produce a different local minimum, and, in general, there does not exist an optimal order that works best in every instance. In practice, the order is determined by efficiency considerations and easily computable operations are used first [1, 2].

Remark While quotienting with ordinary simulation is idempotent, in general this is not true for lookahead simulations, because these relations are not preserved under quotienting (unlike ordinary simulation). Moreover, quotienting w.r.t. forward simulations does not preserve backward simulations, and vice-versa. Our experiments showed that repeatedly and alternatingly quotienting w.r.t. $\preceq^{k\text{-}\mathsf{de}}$ and $\preceq^{k\text{-}\mathsf{bw}}$ (in addition to our pruning techniques) yields the best minimization effect.

The Heavy-k procedure *strictly subsumes* all simulation-based automata minimization methods described in the literature (removing dead states, quotienting, pruning of 'little brother' transitions, mediated preorder), except for the following two: 1) The *fair simulation minimization* of [19] works by tentatively merging fair simulation equivalent states and then checking if this operation preserved the language. (In general, fair simulation is not GFQ.) It subsumes quotienting with $\sqsubseteq^{\mathsf{de}}$ (but not $\preceq^{k\text{-}\mathsf{de}}$) and is implemented in GOAL [34]. We benchmarked our methods against it and found Heavy-k to be much better in both effect and efficiency; cf Section 7. 2) The GFQ *jumping-safe preorders* of [8, 9] are incomparable to the techniques described in this paper. If applied in addition to Heavy-k, they yield a very modest extra minimization effect.

Light-k. This procedure is defined purely for comparison reasons. It demonstrates the effect of the lookahead k in a single quotienting operation and works as follows: Remove all dead states and then quotient w.r.t. $\preceq^{k\text{-}\mathsf{de}}$. Although Light-$k$ achieves much less than Heavy-k, it is not necessarily faster. This is because it uses the more expensive to compute relation $\preceq^{k\text{-}\mathsf{de}}$ directly, while Heavy-k applies other cheaper (pruning) operations first and only then computes $\preceq^{k\text{-}\mathsf{de}}$ on the resulting smaller automaton.

6. Language Inclusion Checking

The language inclusion problem $\mathcal{A} \subseteq \mathcal{B}$ is PSPACE-complete [25]. It can be solved via complementation of $\mathcal{B}$ [31, 34] and, more efficiently, by rank-based ([17] and references therein) or Ramsey-based methods [3, 4, 15, 16], or variants of Piterman's construction [29, 34]. Since these all have *exponential* time complexity, it helps significantly to first minimize the automata in a preprocessing step. Better minimization techniques, as described in the previous sections, make it possible to solve significantly larger instances. However, our simulation-based techniques can not only be used in preprocessing, but actually solve most instances of the inclusion problem *directly*. This is significant, because simulation scales *polynomially* (quadratic average-case complexity; cf. Section 7).

6.1 Inclusion-preserving minimization

Inclusion checking algorithms generally benefit from language-preserving minimization preprocessing (cf. Sec. 5). However, preserving the languages of $\mathcal{A}$ and $\mathcal{B}$ in the preprocessing is not actually necessary. A preprocessing on $\mathcal{A}, \mathcal{B}$ is said to be *inclusion-preserving* iff it produces automata $\mathcal{A}', \mathcal{B}'$ s.t. $\mathcal{A} \subseteq \mathcal{B} \iff \mathcal{A}' \subseteq \mathcal{B}'$

(regardless of whether $\mathcal{A} \approx \mathcal{A}'$ or $\mathcal{B} \approx \mathcal{B}'$). In the following, we consider two inclusion-preserving preprocessing steps.

Simplify $\mathcal{A}$. In theory, the problem $\mathcal{A} \subseteq \mathcal{B}$ is only hard in $\mathcal{B}$, but polynomial in the size of $\mathcal{A}$. However, this is only relevant if one actually constructs the exponential-size complement of $\mathcal{B}$, which is of course to be avoided. For polynomial simulation-based algorithms it is crucial to also minimize $\mathcal{A}$. The idea is to remove transitions in $\mathcal{A}$ which are 'covered' by better transitions in $\mathcal{B}$.

Definition Given $\mathcal{A} = (\Sigma, Q_\mathcal{A}, I_\mathcal{A}, F_\mathcal{A}, \delta_\mathcal{A})$, $\mathcal{B} = (\Sigma, Q_\mathcal{B}, I_\mathcal{B}, F_\mathcal{B}, \delta_\mathcal{B})$, let $P \subseteq \delta_\mathcal{A} \times \delta_\mathcal{B}$ be a relation for comparing transitions in $\mathcal{A}$ and $\mathcal{B}$. The pruned version of $\mathcal{A}$ is $Prune(\mathcal{A}, \mathcal{B}, P) := (\Sigma, Q_\mathcal{A}, I_\mathcal{A}, F_\mathcal{A}, \delta')$ with $\delta' = \{(p,\sigma,r) \in \delta_\mathcal{A} \mid \nexists (p',\sigma',r') \in \delta_\mathcal{B}.\, (p,\sigma,r) P (p',\sigma',r')\}$.

$\mathcal{A} \subseteq \mathcal{B}$ implies $Prune(\mathcal{A}, \mathcal{B}, P) \subseteq \mathcal{B}$ (since $Prune(\mathcal{A}, \mathcal{B}, P) \subseteq \mathcal{A}$). When also the other direction holds (so pruning is inclusion-preserving), we say that P is *good for $\mathcal{A}, \mathcal{B}$-pruning*, i.e., when $\mathcal{A} \subseteq \mathcal{B} \iff Prune(\mathcal{A}, \mathcal{B}, P) \subseteq \mathcal{B}$. Intuitively, pruning is correct when the removed edges do not allow $\mathcal{A}$ to accept any word which is not already accepted by $\mathcal{B}$. In other words, if there is a counter example to inclusion in $\mathcal{A}$, then it can even be found in $Prune(\mathcal{A}, \mathcal{B}, P)$. As in Sec. 3, we compare transitions by looking at their endpoints: For state relations $R_b, R_f \subseteq Q_\mathcal{A} \times Q_\mathcal{B}$, let $P(R_b, R_f) = \{((p,\sigma,r),(p',\sigma,r')) \mid pR_bp' \wedge rR_fr'\}$.

Since inclusion-preserving pruning does not have to respect the language, we can use much weaker (i.e., coarser) relations for comparing endpoints. Let $\subseteq^{\mathsf{bw}-}$ be the variant of $\subseteq^{\mathsf{bw}}$ where accepting states are not taken into consideration.

Theorem 6.1. *$P(\subseteq^{\mathsf{bw}-}, \subseteq^{\mathsf{f}})$ is good for $\mathcal{A}, \mathcal{B}$-pruning.*

Proof. Let $P = P(\subseteq^{\mathsf{bw}-}, \subseteq^{\mathsf{f}})$. One direction is trivial. For the other direction, by contraposition, assume $Prune(\mathcal{A}, \mathcal{B}, P) \subseteq \mathcal{B}$, but $\mathcal{A} \not\subseteq \mathcal{B}$. There exists a $w \in \mathcal{L}(\mathcal{A})$ s.t. $w \notin \mathcal{L}(\mathcal{B})$. There exists an initial fair trace $\pi = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots$ on w in $\mathcal{A}$. There are two cases.

1. π does not contain any transition $q_i \xrightarrow{\sigma_i} q_{i+1}$ that is not present in $Prune(\mathcal{A}, \mathcal{B}, P)$. Then π is also an initial fair trace on w in $Prune(\mathcal{A}, \mathcal{B}, P)$, and thus we obtain $w \in \mathcal{L}(Prune(\mathcal{A}, \mathcal{B}, P))$ and $w \in \mathcal{L}(\mathcal{B})$. Contradiction.
2. π contains a transition $q_i \xrightarrow{\sigma_i} q_{i+1}$ that is not present in $Prune(\mathcal{A}, \mathcal{B}, P)$. Therefore there exists a transition $q'_i \xrightarrow{\sigma_i} q'_{i+1}$ in $\mathcal{B}$ s.t. $q_i \subseteq^{\mathsf{bw}-} q'_i$ and $q_{i+1} \subseteq^{\mathsf{f}} q'_{i+1}$. Thus there exists an initial fair trace on w in $\mathcal{B}$ and thus $w \in \mathcal{L}(\mathcal{B})$. Contradiction. □

We can approximate $\subseteq^{\mathsf{bw}-}$ with (the transitive closure of) a corresponding k-lookahead simulation $\sqsubseteq^{k\text{-}\mathsf{bw}-}$, which is defined as $\sqsubseteq^{k\text{-}\mathsf{bw}}$, except that only initial states are considered, i.e., the winning condition is $\mathcal{C}^{\mathsf{bw}-}(\pi_0, \pi_1) \iff \forall (i \geq 0) \cdot p_i \in I \implies q_i \in I$. Let $\preceq^{k\text{-}\mathsf{bw}-}$ be the transitive closure of $\sqsubseteq^{k\text{-}\mathsf{bw}-}$. Since GFP is $\subseteq$-downward closed and $P(\cdot,\cdot)$ is monotone, we get this corollary.

Corollary 6.2. *$P(\preceq^{k\text{-}\mathsf{bw}-}, \preceq^{k\text{-}\mathsf{f}})$ is good for $\mathcal{A}, \mathcal{B}$-pruning.*

Simplify $\mathcal{B}$. Let $\mathcal{A} \times \mathcal{B}$ be the synchronized product of $\mathcal{A}$ and $\mathcal{B}$. The idea is to remove states in $\mathcal{B}$ which cannot be reached in $\mathcal{A} \times \mathcal{B}$. Let R be the set of states in $\mathcal{A} \times \mathcal{B}$ reachable from $I_\mathcal{A} \times I_\mathcal{B}$, and let $X \subseteq Q_\mathcal{B}$ be the projection of R to the $\mathcal{B}$-component. We obtain $\mathcal{B}'$ from $\mathcal{B}$ by removing all states $\notin X$ and their associated transitions. Although $\mathcal{B}' \not\approx \mathcal{B}$, this operation is clearly inclusion-preserving.

6.2 Jumping fair simulation as a better GFI relation

We further generalize the GFI preorder $\preceq^{k\text{-}\mathsf{f}}$ by allowing Duplicator even more freedom. The idea is to allow Duplicator to take *jumps* during the simulation game (in the spirit of [9]). For a preorder $\leq$ on Q, in the game for *$\leq$-jumping k-lookahead simulation* Duplicator is allowed to jump to $\leq$-larger states before taking a transition. Thus, a Duplicator's move is of the form $q_i \leq q'_i \xrightarrow{\sigma_i} q_{i+1} \leq q'_{i+1} \xrightarrow{\sigma_{i+1}} \cdots \xrightarrow{\sigma_{i+m-1}} q_{i+m}$, and she eventually builds an infinite $\leq$-jumping trace. We say that this trace is *accepting* at step i iff $\exists q''_i \in F.\, q_i \leq q''_i \leq q'_i$, and *fair* iff it is accepting infinitely often. As usual, *$\leq$-jumping k-lookahead fair simulation* holds iff Duplicator wins the corresponding game, with the fair winning condition lifted to jumping traces.

Not all preorders $\leq$ induce GFI jumping simulations. The preorder $\leq$ is called *jumping-safe* [9] if, for every word w, there exists a $\leq$-jumping initial fair trace on w iff there exists an initial fair non-jumping one. Thus, jumping-safe preorders allows to convert jumping traces into non-jumping ones. Consequently, for a jumping-safe preorder $\leq$, $\leq$-jumping k-lookahead fair simulation is GFI.

One can prove that $\subseteq^{\mathsf{bw}}$ is jumping-safe, while $\subseteq^{\mathsf{bw}-}$ is not. We even improve $\subseteq^{\mathsf{bw}}$ to a slightly more general jumping-safe relation $\subseteq^{\mathsf{bw\text{-}c}}$, by only requiring that Duplicator visits at least as many accepting states as Spoiler does, but not necessarily at the same time. Formally, $p_m \subseteq^{\mathsf{bw\text{-}c}} q_m$ iff, for every initial w-trace $\pi_0 = p_0 \xrightarrow{\sigma_0} p_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{m-1}} p_m$, there exists an initial w-trace $\pi_1 = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{m-1}} q_m$, s.t. $|\{i \mid p_i \in F\}| \leq |\{i \mid q_i \in F\}|$.

Theorem 6.3. *The preorder $\subseteq^{\mathsf{bw\text{-}c}}$ is jumping-safe.*

Proof. Since $\subseteq^{\mathsf{bw\text{-}c}}$ is reflexive, the existence of an initial fair trace on w directly implies the existence of a $\subseteq^{\mathsf{bw\text{-}c}}$-jumping initial fair trace on w.

Now, we show the reverse implication. Given two initial $\subseteq^{\mathsf{bw\text{-}c}}$-jumping traces on w $\pi_0 = p_0 \subseteq^{\mathsf{bw\text{-}c}} p'_0 \xrightarrow{\sigma_0} p_1 \subseteq^{\mathsf{bw\text{-}c}} p'_1 \xrightarrow{\sigma_1} \cdots$ and $\pi_1 = q_0 \subseteq^{\mathsf{bw\text{-}c}} q'_0 \xrightarrow{\sigma_0} q_1 \subseteq^{\mathsf{bw\text{-}c}} q'_1 \xrightarrow{\sigma_1} \cdots$ we define $\mathcal{C}^c_j(\pi_0, \pi_1)$ iff $|\{i \leq j \mid \exists p''_i \in F.\, p_i \subseteq^{\mathsf{bw\text{-}c}} p''_i \subseteq^{\mathsf{bw\text{-}c}} p'_i\}| \leq |\{i \leq j \mid \exists q''_i \in F.\, q_i \subseteq^{\mathsf{bw\text{-}c}} q''_i \subseteq^{\mathsf{bw\text{-}c}} q'_i\}|$. We say that an initial $\subseteq^{\mathsf{bw\text{-}c}}$-jumping trace on w is *i-good* iff it does not jump within the first i steps.

We show, by induction on i, the following property (P): For every i and every infinite $\subseteq^{\mathsf{bw\text{-}c}}$-jumping initial trace $\pi = p_0 \subseteq^{\mathsf{bw\text{-}c}} p'_0 \xrightarrow{\sigma_0} p_1 \subseteq^{\mathsf{bw\text{-}c}} p'_1 \xrightarrow{\sigma_1} \cdots$ on w there exists an initial i-good trace $\pi^i = q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_i} q_i \cdots$ on w s.t. $\mathcal{C}^c_i(\pi, \pi^i)$ and the suffixes of the traces are identical, i.e., $q_i = p_i$ and $\pi[i..] = \pi^i[i..]$.

For the case base $i = 0$ we take $\pi^0 = \pi$. Now we consider the induction step. By induction hypothesis we get an initial i-good trace π^i s.t. $\mathcal{C}^c_i(\pi, \pi^i)$ and $q_i = p_i$ and $\pi[i..] = \pi^i[i..]$. If π^i is $(i+1)$-good then we can take $\pi^{i+1} = \pi^i$. Otherwise, π^i contains a step $q_i \subseteq^{\mathsf{bw\text{-}c}} q'_i \xrightarrow{\sigma_i} q_{i+1}$. First we consider the case where there exists a $q''_i \in F$ s.t. $q_i \subseteq^{\mathsf{bw\text{-}c}} q''_i \subseteq^{\mathsf{bw\text{-}c}} q'_i$. (Note that the i-th step in π^i can count as accepting in $\mathcal{C}^c$ because $q''_i \in F$, even if q_i and q'_i are not accepting.) By def. of $\subseteq^{\mathsf{bw\text{-}c}}$ there exists an initial trace π'' on a prefix of w that ends in q''_i and visits accepting states at least as often as the non-jumping prefix of π^i that ends in q_i. Again by definition of $\subseteq^{\mathsf{bw\text{-}c}}$ there exists an initial trace π' on a prefix of w that ends in q'_i and visits accepting states at least as often as π''. Thus π' visits accepting states at least as often as the *jumping* prefix of π^i that ends in q'_i (by the definition of $\mathcal{C}^c$). By composing the traces we get $\pi^{i+1} = \pi'(q'_i \xrightarrow{\sigma_i} q_{i+1})\pi^i[i+1..]$. Thus π^{i+1} is an $(i+1)$-good initial trace on w and $\pi[i+1..] = \pi^i[i+1..] = \pi^{i+1}[i+1..]$ and $\mathcal{C}^c_{i+1}(\pi^i, \pi^{i+1})$ and $\mathcal{C}^c_{i+1}(\pi, \pi^{i+1})$. The other case where there is no $q''_i \in F$ s.t. $q_i \subseteq^{\mathsf{bw\text{-}c}} q''_i \subseteq^{\mathsf{bw\text{-}c}} q'_i$ is similar, but simpler.

Let π be an initial $\subseteq^{\mathsf{bw\text{-}c}}$-jumping fair trace on w. By property (P) and König's Lemma there exists an infinite initial non-jumping fair trace π' on w. Thus $\subseteq^{\mathsf{bw\text{-}c}}$ is jumping-safe. □

As a direct consequence, $\subseteq^{\text{bw-c}}$-jumping k-lookahead fair simulation is GFI. Since $\subseteq^{\text{bw-c}}$ is difficult to compute, we approximate it by a corresponding lookahead-simulation $\sqsubseteq^{k\text{-bw-c}}$ which, in the same spirit, counts and compares the number of visits to accepting states in every round of the k-lookahead backward simulation game. Let $\preceq^{k\text{-bw-c}}$ be the transitive closure of $\sqsubseteq^{k\text{-bw-c}}$.

Corollary 6.4. *$\preceq^{k\text{-bw-c}}$-jumping k-lookahead fair sim. is GFI.*

6.3 Advanced inclusion checking algorithm

Given these techniques, we propose the following algorithm for inclusion checking $\mathcal{A} \subseteq \mathcal{B}$.

(1) Use the Heavy-k procedure to minimize $\mathcal{A}$ and $\mathcal{B}$, and additionally apply the inclusion-preserving minimization techniques from Sec. 6. Lookahead simulations are computed not only on $\mathcal{A}$ and $\mathcal{B}$, but also *between* them (i.e., on their disjoint union). Since they are GFI, we check whether they already witness inclusion. Since many simulations are computed between partly minimized versions of $\mathcal{A}$ and $\mathcal{B}$, this witnesses inclusion much more often than checking fair simulation between the original versions. This step either stops showing inclusion, or produces smaller inclusion-equivalent automata $\mathcal{A}', \mathcal{B}'$.

(2) Check the GFI $\preceq^{k\text{-bw-c}}$-jumping k-lookahead fair simulation from Sec. 6.2 between $\mathcal{A}'$ and $\mathcal{B}'$, and stop if the answer is yes.

(3) If inclusion was not established in steps (1) or (2) then try to find a counterexample to inclusion. This is best done by a Ramsey-based method (optionally using simulation-based subsumption techniques), e.g., [1, 4]. Use a small timeout value, since in most non-included instances there exists a very short counterexample. Stop if a counterexample is found.

(4) If steps (1)-(3) failed (rare in practice), use any complete method, (e.g., Rank-based, Ramsey-based or Piterman's construction) to check $\mathcal{A}' \subseteq \mathcal{B}'$. At least, it will benefit from working on the smaller instance $\mathcal{A}', \mathcal{B}'$ produced by step (1).

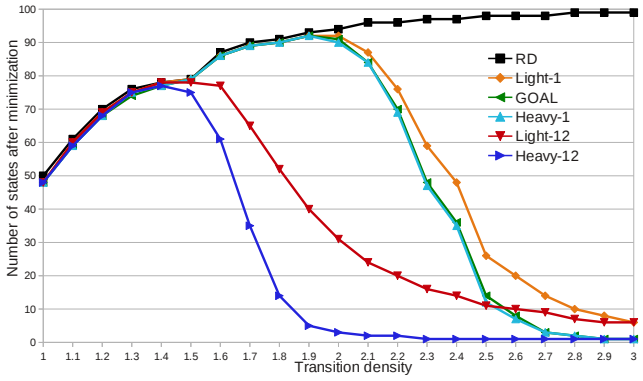
Note that steps (1)-(3) take polynomial time, while step (4) takes exponential time. (For the latter, we recommend the improved Ramsey method of [1, 4] and the on-the-fly variant of Piterman's construction [29] implemented in GOAL [34].) This algorithm allows to solve much larger instances of the inclusion problem than previous methods [3, 4, 15–17, 29, 31, 34], i.e., automata with 1000-20000 states instead of 10-100 states; cf. Section 7.

7. Experiments

We test the effectiveness of Heavy-k minimization on Tabakov-Vardi random automata [33], on automata derived from LTL formulae, and on automata derived from mutual exclusion protocols, and compare it to the best previously available techniques implemented in GOAL [34]. A scalability test shows that Heavy-k has quadratic average-case complexity and it is vastly more efficient than GOAL. Furthermore, we test our methods for language inclusion on large instances and compare their performance to previous techniques. Due to space limitations, we only give a summary of the results, but all details and the runnable tools are available [2]. Unless otherwise stated, the experiments were run with Java 6 on Intel Xeon X5550 2.67GHz and 14GB memory.

Random automata. The Tabakov-Vardi model [33] generates random automata according to the following parameters: the number of states n, the size of the alphabet $|\Sigma|$, the transition density td (number of transitions, relative to n and $|\Sigma|$) and the acceptance density ad (percentage of accepting states). Apart from this, they do not have any special structure, and thus minimization and language inclusion problem are harder for them than for automata from other sources (see below). Random automata provide general reproducible test cases, on average. Moreover, they are the only test cases that are guaranteed to be unbiased towards any particular method. Thus, it is a particular sign of quality if a method performs well even on these hard cases.

The inherent difficulty of the minimization problem, and thus also the effectiveness of minimization methods, depends strongly on the class of random automata, i.e., on the parameters listed above. Thus, one needs to compare the methods over the whole range, not just for one example. Variations in ad do not affect Heavy-k much (cf. Appendix A.1), but very small values make minimization harder for the other methods. By far the most important parameter is td. The following figure shows typical results. We take $n = 100$, $|\Sigma| = 2$, $ad = 0.5$ and the range of $td = 1.0, 1.1, \ldots, 3.0$. For each td we created 300 random automata, minimized them with different methods, and plotted the resulting average number of states after minimization. Each curve represents a different method: RD (just remove dead states), Light-1, Light-12, Heavy-1, and Heavy-12 and GOAL. The GOAL curve shows the best effort of all previous techniques (as implemented in GOAL), which include RD, quotienting with backward and forward simulation, pruning of little brother transitions and the fair simulation minimization of [19] (which subsumes quotienting with delayed simulation).



Sparse automata with low td have more dead states. For $td \leq 1.4$ no technique except RD has any significant effect. GOAL minimizes just slightly worse than Heavy-1 but it is no match for our best techniques. Heavy-12 vastly outperforms all others, particularly in the interesting range between 1.4 and 2.5. Moreover, the minimization of GOAL (in particular the fair simulation minimization of [19]) is very slow. For GOAL, the average minimization time per automaton varies between 39s (at $td = 1.0$) and 612s (maximal at $td = 2.9$). In contrast, for Heavy-12, the average minimization time per automaton varies between 0.012s (at $td = 1.0$) and 1.482s (max. at $td = 1.7$). So Heavy-12 minimizes not only much better, but also at least 400 times faster than GOAL (see also the scalability test).

For $td \geq 2.0$, Heavy-12 yields very small automata. Many of these are even universal, i.e., with just one state and a universal loop. However, this frequent universality is *not* due to trivial reasons (otherwise simpler techniques like Light-1 and GOAL would also recognize this). Consider the following question: Given Tabakov-Vardi random automata with parameters n, $|\Sigma|$ and td, what is the probability $U(n, |\Sigma|, td)$ that every state has at least one outgoing transition for every symbol in Σ? (Such an automaton would be trivially universal if $ad = 1$.)

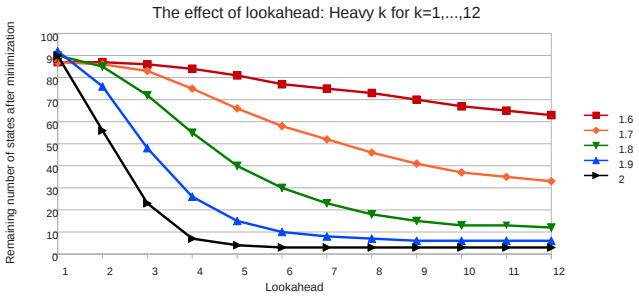
Theorem 7.1. *$U(n,|\Sigma|,td) = (\alpha(n,T)/\beta(n,T))^{|\Sigma|}$, with $T = n \cdot td$, $\alpha(n,T) = \sum_{m=n}^{n^2} \binom{m-n}{T-n} \sum_{i=0}^{n} (-1)^i \binom{n}{i} \binom{m-in-1}{n-1}$ and $\beta(n,T) = \binom{n^2}{T}$*

Proof. For each symbol in Σ there are $T = n \cdot td$ transitions and n^2 possible places for transitions, described as a grid. $\alpha(n,T)$ is

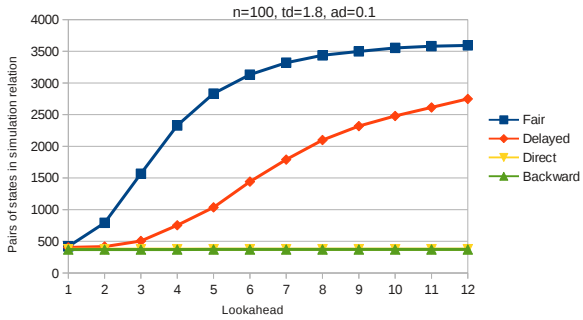
the number of ways T items can be placed onto an $n \times n$ grid s.t. every row contains ≥ 1 item, i.e., every state has an outgoing transition. $\beta(n,T)$ is the number of possibilities without this restriction, which is trivially $\binom{n^2}{T}$. Since the Tabakov-Vardi model chooses transitions for different symbols independently, we have $U(n,|\Sigma|,td) = (\alpha(n,T)/\beta(n,T))^{|\Sigma|}$. It remains to compute $\alpha(n,T)$. For the i-th row let $x_i \in \{1,\ldots,n\}$ be the maximal column containing an item. The remaining $T-n$ items can only be distributed to lower columns. Thus $\alpha(n,T) = \sum_{x_1,\ldots,x_n} \binom{(\sum x_i)-n}{T-n}$. With $m = \sum x_i$ and a standard dice-sum problem from [28] the result follows. □

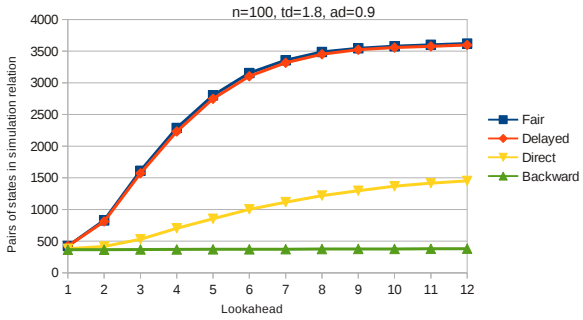
For $n = 100$, $|\Sigma| = 2$ we obtain the following values for $U(n,|\Sigma|,td)$: 10^{-15} for $td = 2.0$, $2.9 \cdot 10^{-5}$ for $td = 3.0$, 0.03 for $td = 4.0$, 0.3 for $td = 5.0$, 0.67 for $td = 6.0$, and 0.95 for $td = 8.0$. So this transition saturation effect is negligible in our tested range with $td \leq 3.0$.

While Heavy-12 performs very well, an even smaller lookahead can already be sufficient for a good minimization. However, this depends very much on the density td of the automata. The following chart shows the effect of the lookahead by comparing Heavy-k for varying k on different classes of random automata with different density $td = 1.6, 1.7, 1.8, 1.9, 2.0$. We have $n = 100$, $|\Sigma| = 2$ and $ad = 0.5$, and every point is the average of 1000 automata.



The big advantage of Heavy-12 over Light-12 is due to the pruning techniques. However, these only reach their full potential at higher lookaheads (thus the smaller difference between Heavy-1 and Light-1). Indeed, the simulation relations get much denser with higher lookahead k. We consider random automata with $n = 100$, $|\Sigma| = 2$ and $td = 1.8$ (a nontrivial case; larger td yield larger simulations). We let $ad = 0.1$ (resp. $ad = 0.9$), and plot the size of fair, delayed, direct, and backward simulation as k increases from 1 to 12. Every point is the average of 1000 automata.





Fair/delayed simulation is not much larger than direct simulation for $k = 1$, but they benefit strongly from higher k. Backward simulation increases only slightly (e.g., from 365 to 381 pairs for $ad = 0.9$). Initially, it seems as if backward/direct simulation does not benefit from higher k if ad is small (on random automata), but this is wrong. Even random automata get less random during the Heavy-k minimization process, making lookahead more effective for backward/direct simulation. Consider the case of $n = 300$, $td = 1.8$ and $ad = 0.1$. Initially, the average ratio $|\preceq^{12\text{-di}}|/|\preceq^{1\text{-di}}|$ is 1.00036, but after quotienting with $\preceq^{12\text{-de}}$ this ratio is 1.103.

LTL. For model checking [22], LTL-formulae are converted into Büchi automata. This conversion has been extensively studied and there are many different algorithms which try to construct the smallest possible automaton for a given formula (see references in [34]). It should be noted however, that LTL is designed for human readability and does not cover the full class of ω-regular languages. Moreover, Büchi Store [35] contains handcrafted automata for almost every human-readable LTL-formula and none of these automata has more than 7 states. Still, since many people are interested in LTL to automata conversion, we tested how much our minimization algorithm can improve upon the best effort of previous techniques. For LTL model checking, the size of the automata is not the only criterion [30], since more non-determinism also makes the problem harder. However, our transition pruning techniques only make an automaton 'more deterministic'.

Using a function of GOAL, we created 300 random LTL-formulae of nontrivial size: length 70, 4 predicates and probability weights 1 for boolean and 2 for future operators. We then converted these formulae to Büchi automata and minimized them with GOAL. Of the 14 different converters implemented in GOAL we chose LTL2BA [18] (which is also used by the SPIN model checker [22]), since it was the only one which could handle such large formulae. (The second best was COUVREUR which succeeded on 90% of the instances, but produced much larger automata than LTL2BA. The other converters ran out of time (4h) or memory (14GB) on most instances.) We thus obtained 300 automata and minimized them with GOAL. The resulting automata vary significantly in size from 1 state to 1722 states [2].

Then we tested how much *further* these automata could be reduced in size by our Heavy-12 method (cf. Appendix A.2). In summary, 82% of the automata could be further reduced in size. The average number of states declined from 138 to 78, and the average number of transitions from 3102 to 1270. Since larger automata have a disproportionate effect on averages, we also computed the average reduction ratio per automaton, i.e., $(1/300)\sum_{i=1}^{300} newsize_i/oldsize_i$. (Note the difference between the average ratio and the ratio of averages.) The average ratio was 0.76 for states and 0.68 for transitions. The computation times for minimization vary a lot due to different automata sizes (average 122s), but were always less than the time used by the LTL to automata translation. If one only considers the 150 automata above median size (30 states) then the results are even stronger. 100% of these

automata could be further reduced in size. The average number of states declined from 267 to 149, and the average number of transitions from 6068 to 2435. The average reduction ratio was 0.65 for states and 0.54 for transitions. To conclude, our minimization can significantly improve the quality of LTL to automata translation with a moderate overhead.

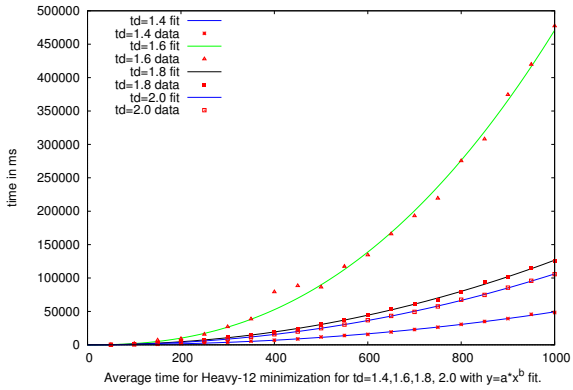
Mutual exclusion protocols. We consider automata derived from mutual exclusion protocols. The protocols were described in a language of guarded commands and automatically translated into Büchi automata, whose size is given in the column 'Original'. By row, the protocols are Bakery.1, Bakery.2, Fischer.3.1, Fischer.3.2, Fischer.2, Phils.1.1, Phils.2 and Mcs.1.2. We minimize these automata with GOAL and with our Heavy-12 method and describe the sizes of the resulting automata and the runtime in subsequent columns (Java 6 on Intel i7-740, 1.73 GHz). In some instances GOAL ran out of time (2h) or memory (14GB).

Original		GOAL		Time	Heavy-12		Time
Trans.	States	Tr.	St.	GOAL	Tr.	St.	Heavy-12
2597	1506	N/A	N/A	$> 2h$	696	477	6.17s
2085	1146	N/A	N/A	$> 2h$	927	643	9.04s
1401	638	14	10	15.38s	14	10	1.16s
3856	1536	212	140	4529s	96	70	5.91s
67590	21733	N/A	N/A	oom(14GB)	316	192	325.76s
464	161	362	134	540.3s	359	134	11.51s
2350	581	284	100	164.2s	225	97	4.04s
21509	7968	108	69	2606.7s	95	62	48.18s

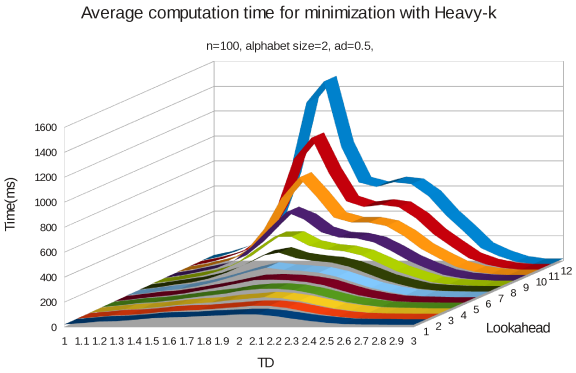
Scalability. We test the scalability of Heavy-12 minimization by applying it to Tabakov-Vardi random automata of increasing size but fixed *td*, *ad* and Σ. We ran four separate tests with $td = 1.4, 1.6, 1.8$ and 2.0. In each test we fixed $ad = 0.5$, $|\Sigma| = 2$ and increased the number of states from $n = 50$ to $n = 1000$ in increments of 50. For each parameter point we created 300 random automata and minimized them with Heavy-12. We analyze the average size of the minimized automata in percent of the original size n, and how the average computation time increases with n.

For $td = 1.4$ the average size of the minimized automata stays around 77% of the original size, regardless of n. For $td = 1.6$ it stays around 65%. For $td = 1.8$ it *decreases* from 28% at $n = 50$ to 2% at $n = 1000$. For $td = 2.0$ it *decreases* from 8% at $n = 50$ to $< 1\%$ at $n = 1000$ (cf. Appendix A.2). Note that the lookahead of 12 did *not change* with n. Surprisingly, larger automata do not require larger lookahead for a good minimization.

We plot the average computation time (measured in ms) in n and then compute the optimal fit of the function $time = a * n^b$ to the data by the least-squares method, i.e., this computes the parameters a and b of the function that most closely fits the experimental data. The important parameter is the exponent b. For $td = 1.4, 1.6, 1.8, 2.0$ we obtain $0.018 * n^{2.14}$, $0.32 * n^{2.39}$, $0.087 * n^{2.05}$ and $0.055 * n^{2.09}$, respectively. Thus, the average-case complexity of Heavy-12 scales (almost) quadratically. This is especially surprising given that Heavy-12 does not only compute one simulation relation but potentially many simulations until the repeated minimization reaches a fixpoint. Quadratic complexity is the very best one can hope for in any method that explicitly compares states/transitions by simulation relations, since the relations themselves are of quadratic size. Lower complexity is only possible with pure partition refinement techniques (e.g., bisimulation, which is $O(n \log n)$), but these achieve even less minimization than quotienting with direct simulation (i.e., next to nothing on hard instances).



The computation time of Heavy-k depends on the class of automata, i.e., on the density *td*, as the scalability test above shows. Moreover, it also depends on *k*. The following graph shows the average computation time of Heavy-k on automata of size 100 and varying *td* and *k*. The most difficult cases are those where minimization is possible (and thus the alg. does not give up quickly), but does not massively reduce the size of the instance. For Heavy-k, this peak is around $td = 1.6, 1.7$ (like in the scalability test).



Language Inclusion Checking. We test the language inclusion checking algorithm of Section 6.3 (with lookahead up-to 15) on nontrivial instances and compare its performance to previous techniques like ordinary fair simulation checking and the best effort of GOAL (which uses simulation-based minimization followed by an on-the-fly variant of Piterman's construction [29, 34]). In this test we use only the polynomial time steps (1)-(3) of our algorithm, thus it may fail in some instances. We consider pairs of Tabakov-Vardi random automata with 1000 states each, $|\Sigma| = 2$ and $ad = 0.5$. For each separate case of $td = 1.6, 1.8$ and 2.0, we create 300 such automata pairs and check if language inclusion holds. (For $td < 1.6$ inclusion rarely holds, except trivially if one automaton has empty language. For $td > 2$ inclusion often holds but is easier to prove.)

For $td = 1.6$ our algorithm solved 294 of 300 instances (i.e., 98%): 45 included (16 in step (1) and 29 in step (2)), 249 non-included (step (3)), and 6 failed. Average computation time 1167s. Ordinary fair simulation solved only 13 included instances. GOAL (timeout 60min, 14GB memory) solved only 13 included instances (the same 13 as fair simulation) and 155 non-included instances.

For $td = 1.8$ our algorithm solved 297 of 300 instances (i.e., 99%): 104 included (103 in step (1) and 1 in step (2)) and 193 non-included (step (3)) and 3 failed. Average computation time 452s. Ordinary fair simulation solved only 5 included instances. GOAL (timeout 30min, 14GB memory) solved only 5 included instances (the same 5 as fair simulation) and 115 non-included instances.

For $td = 2.0$ our algorithm solved every instance: 143 included (shown in step (1)) and 157 non-included (step (3)). Average com-

putation time 258s. Ordinary fair simulation solved only 1 of the 143 included instances. GOAL (timeout 30min, 14GB memory) solved only 1 of 143 included instances (the same one as fair simulation) and 106 of 157 non-included instances.

8. Conclusion and Future Work

Our automata minimization techniques perform significantly better than previous methods. In particular, they can be applied to solve PSPACE-complete automata problems like language inclusion for much larger instances. While we presented our methods in the context of Büchi automata, most of them trivially carry over to the simpler case of automata over finite words. Future work includes more efficient algorithms for computing lookahead simulations, either along the lines of [20] for normal simulation, or by using symbolic representations of the relations. Moreover, we are applying similar techniques to minimize tree-automata.

References

[1] RABIT tool: www.languageinclusion.org/doku.php?id=tools.

[2] See the appendix of this technical report for details, and www.languageinclusion.org/doku.php?id=tools for the Java code of the tools and the data of the experiments.

[3] P. Abdulla, Y.-F. Chen, L. Clemente, L. Holik, C.-D. Hong, R. Mayr, and T. Vojnar. Simulation Subsumption in Ramsey-Based Büchi Automata Universality and Inclusion Testing. In T. Touili, B. Cook, and P. Jackson, editors, *Computer Aided Verification*, volume 6174 of *LNCS*, pages 132–147, 2010. ISBN 978-3-642-14294-9. doi: 10.1007/978-3-642-14295-6_14. URL http://dx.doi.org/10.1007/978-3-642-14295-6_14.

[4] P. Abdulla, Y.-F. Chen, L. Clemente, L. Holik, C.-D. Hong, R. Mayr, and T. Vojnar. Advanced Ramsey-based Büchi Automata Inclusion Testing. In J.-P. Katoen and B. König, editors, *International Conference on Concurrency Theory*, volume 6901 of *LNCS*, pages 187–202, Sept. 2011.

[5] P. A. Abdulla, Y.-F. Chen, L. Holík, and T. Vojnar. Mediating for reduction (on minimizing alternating Büchi automata). In *FSTTCS*, volume 4 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2009.

[6] P. A. Abdulla, Y.-F. Chen, L. Holik, R. Mayr, and T. Vojnar. When Simulation Meets Antichains. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *LNCS*, 2010. URL http://hal.inria.fr/inria-00460294/en/.

[7] D. Bustan and O. Grumberg. Simulation-based minimization. *ACM Trans. Comput. Logic*, 4:181–206, April 2003. ISSN 1529-3785. doi: http://doi.acm.org/10.1145/635499.635502. URL http://doi.acm.org/10.1145/635499.635502.

[8] L. Clemente. Büchi Automata Can Have Smaller Quotients. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP*, volume 6756 of *LNCS*, pages 258–270. 2011. ISBN 978-3-642-22011-1. doi: 10.1007/978-3-642-22012-8_20. URL http://arxiv.org/pdf/1102.3285.

[9] L. Clemente. *Generalized Simulation Relations with Applications in Automata Theory*. PhD thesis, University of Edinburgh, 2012.

[10] D. L. Dill, A. J. Hu, and H. Wont-Toi. Checking for Language Inclusion Using Simulation Preorders. In *Computer Aided Verification*, volume 575 of *LNCS*. Springer-Verlag, 1991. doi: 10.1007/3-540-55179-4_25. URL http://dx.doi.org/10.1007/3-540-55179-4_25.

[11] L. Doyen and J.-F. Raskin. Antichains Algorithms for Finite Automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *LNCS*, pages 2–22. Springer-Verlag, 2010.

[12] K. Etessami. A Hierarchy of Polynomial-Time Computable Simulations for Automata. In *International Conference on Concurrency Theory*, volume 2421 of *LNCS*, pages 131–144. Springer-Verlag, 2002. doi: 10.1007/3-540-45694-5_10. URL http://dx.doi.org/10.1007/3-540-45694-5_10.

[13] K. Etessami and G. Holzmann. Optimizing Büchi Automata. In *International Conference on Concurrency Theory*, volume 1877 of *LNCS*, pages 153–168. Springer-Verlag, 2000.

[14] K. Etessami, T. Wilke, and R. A. Schuller. Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata. *SIAM J. Comput.*, 34(5):1159–1175, 2005. doi: 10.1137/S0097539703420675. URL http://epubs.siam.org/sam-bin/dbq/article/42067.

[15] S. Fogarty and M. Vardi. Büchi Complementation and Size-Change Termination. In S. Kowalewski and A. Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 5505 of *LNCS*, pages 16–30. 2009. doi: 10.1007/978-3-642-00768-2_2. URL http://dx.doi.org/10.1007/978-3-642-00768-2_2.

[16] S. Fogarty and M. Y. Vardi. Efficient Büchi Universality Checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 205–220, 2010.

[17] S. Fogarty, O. Kupferman, M. Y. Vardi, and T. Wilke. Unifying Büchi Complementation Constructions. In M. Bezem, editor, *Computer Science Logic*, volume 12 of *LIPIcs*, pages 248–263. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011. doi: http://dx.doi.org/10.4230/LIPIcs.CSL.2011.248.

[18] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *CAV*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001.

[19] S. Gurumurthy, R. Bloem, , and F. Somenzi. Fair simulation minimization. In *CAV*, volume 2404 of *LNCS*, pages 610–624. Springer, 2002.

[20] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *Foundations of Computer Science*, FOCS '95, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7183-1. URL http://portal.acm.org/citation.cfm?id=796255.

[21] T. A. Henzinger, O. Kupferman, and S. K. Rajamani. Fair Simulation. *Information and Computation*, 173:64–81, 2002. doi: 10.1006/inco.2001.3085. URL http://dx.doi.org/10.1006/inco.2001.3085.

[22] G. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2004.

[23] T. Jiang and B. Ravikumar. Minimal NFA Problems are Hard. In J. Albert, B. Monien, and M. Artalejo, editors, *ICALP*, volume 510 of *LNCS*, pages 629–640. 1991. doi: 10.1007/3-540-54233-7_169.

[24] S. Juvekar and N. Piterman. Minimizing Generalized Büchi Automata. In *Computer Aided Verification*, volume 4414 of *LNCS*, pages 45–58. Springer-Verlag, 2006. doi: 10.1007/11817963_7. URL http://dx.doi.org/10.1007/11817963_7.

[25] O. Kupferman and M. Vardi. Verification of Fair Transition Systems. In *Computer Aided Verification*, volume 1102 of *LNCS*, pages 372–382. Springer-Verlag, 1996. URL http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.9654.

[26] C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. POPL '01, pages 81–92, 2001. doi: http://doi.acm.org/10.1145/360204.360210.

[27] J. Leroux and G. Point. TaPAS: The Talence Presburger Arithmetic Suite. In *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 5505 of *LNCS*. Springer, 2009.

[28] I. Niven. *Mathematics of Choice*. The Mathematical Association of America, 1965.

[29] N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264. IEEE, 2006.

[30] R. Sebastiani and S. Tonetta. More deterministic vs. smaller Büchi automata for efficient LTL model checking. In *Correct Hardware Design and Verification Methods*, volume 2860 of *LNCS*, 2003.

[31] A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theor. Comput. Sci.*, 49:217–237, Jan. 1987. ISSN 0304-3975. doi: 10.1016/0304-3975(87)90008-9. URL http://dx.doi.org/10.1016/0304-3975(87)90008-9.

[32] F. Somenzi and R. Bloem. Efficient Büchi Automata from LTL Formulae. In *Computer Aided Verification*, volume 1855 of *LNCS*, pages 248–263. Springer-Verlag, 2000. doi: 10.1007/10722167_21. URL http://dx.doi.org/10.1007/10722167_21.

[33] D. Tabakov and M. Vardi. Model Checking Büchi Specifications. In *LATA*, volume Report 35/07. Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona, 2007.

[34] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, W.-C. Chan, and C.-J. Luo. GOAL extended: Towards a research tool for omega automata and temporal logic. In C. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *LNCS*, pages 346–350. 2008. ISBN 978-3-540-78799-0. URL http://dx.doi.org/10.1007/978-3-540-78800-3_26.

[35] Y.-K. Tsay, M.-H. Tsai, J.-S. Chang, and Y.-W. Chang. Büchi store: An open repository of Büchi automata. In P. Abdulla and K. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *LNCS*, pages 262–266. 2011. ISBN 978-3-642-19834-2. URL http://dx.doi.org/10.1007/978-3-642-19835-9_23. 10.1007/978-3-642-19835-9_23.

A. Additional Experiments

This appendix contains additional material related to experiments with our minimization algorithm (cf. Section 7).

A.1 The Effect of the Acceptance Density

Figure 3 shows the performance of our minimization algorithm on random automata with acceptance density 0.5 and 0.1, respectively. Clearly, variations in the acceptance density do not affect our methods with lookahead (e.g., Light-12 and Heavy-12) very much. However, a small acceptance density like 0.1 makes the problem somewhat harder for methods without lookahead (e.g., Light-1 and Heavy-1).

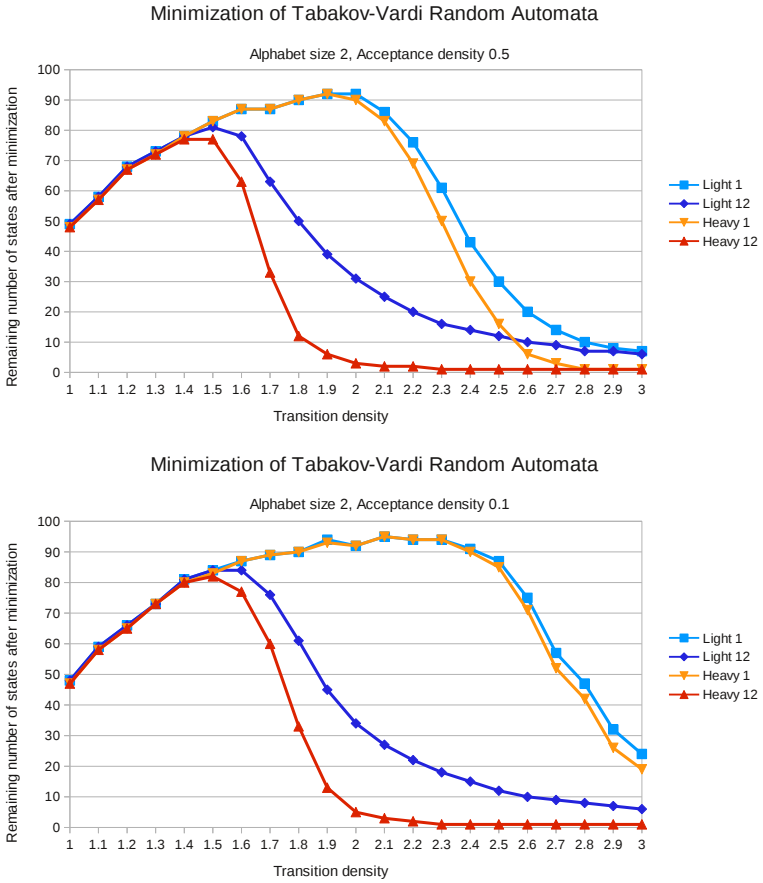


Figure 3. Minimization of Tabakov-Vardi random automata with $n = 100$, $|\Sigma| = 2$, $ad = 0.5$ (top), $ad = 0.1$ (bottom) and varying td. We use the Light 1, Light 12, Heavy 1 and Heavy 12 methods and plot the average number of states of the minimized automata. Every point in the top (resp. bottom) graph the average of 1000 (resp. 300) automata. Note how a small acceptance density makes minimization harder without lookahead, but not much harder for lookahead 12.

A.2 Scalability

In this section we present the complete data for our scalability experiments. We tested our Heavy-12 minimization algorithm on random automata of increasing size but fixed *td*, *ad* and Σ. In Figure 4 we show the reduction in size, while in Figure 5 we show the computation time (for the same set of experiments).

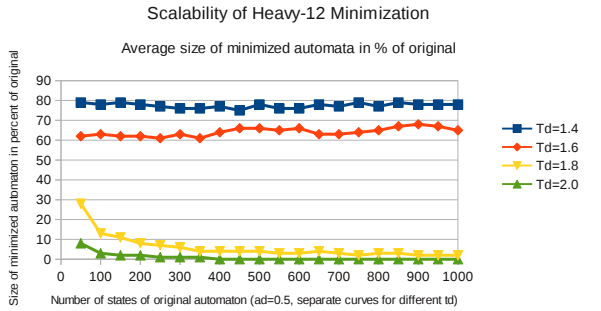


Figure 4. Minimization of Tabakov-Vardi random automata with $ad = 0.5$, $|\Sigma| = 2$, and increasing $n = 50, 100, \ldots, 1000$. Different curves for different *td*. We plot the average size of the Heavy-12 minimized automata, in percent of their original size. Every point is the average of 300 automata. Note that the lookahead of 12 does not change, i.e., larger automata do not require a higher lookahead for a good minimization.

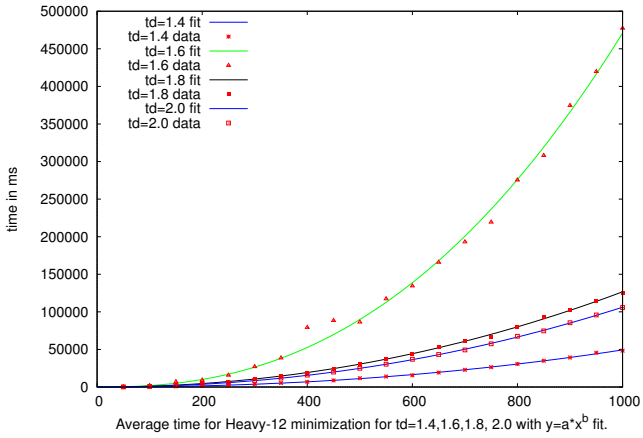


Figure 5. Minimization of Tabakov-Vardi random automata as in Figure 4. Here we plot the average computation time (in ms) for the minimization, and a least-squares fit by the function $a * n^b$. For $td = 1.4, 1.6, 1.8, 2.0$ we obtain $0.018 * n^{2.14}$, $0.32 * n^{2.39}$, $0.087 * n^{2.05}$ and $0.055 * n^{2.09}$, respectively.

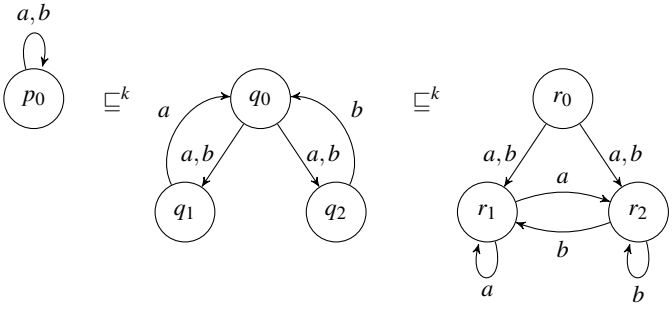


Figure 6. Lookahead simulation is not transitive.

B. Non-transitivity of Lookahead Simulation

In this section we show that lookahead simulation is not transitive for $k \geq 2$. Consider the example in Figure 6. We have $p_0 \sqsubseteq^k q_0 \sqsubseteq^k r_0$ (and $k = 2$ suffices), but $p_0 \not\sqsubseteq^k r_0$ for any $k > 0$. In fact,

- $p_0 \sqsubseteq^k q_0$, with $k = 2$: Duplicator takes the transition via q_1 or q_2 depending on whether Spoiler plays word $(a+b)a$ or $(a+b)b$, respectively.
- $q_0 \sqsubseteq^k r_0$, with $k = 2$: If Spoiler goes to q_1 or q_2, then Duplicator goes to r_1 or r_2, respectively. That $q_1 \sqsubseteq^k r_1$ holds can be shown as follows (the case $q_2 \sqsubseteq^k r_2$ is similar). If Spoiler takes transitions $q_1 \xrightarrow{a} q_0 \xrightarrow{a} q_1$, then Duplicator does $r_1 \xrightarrow{a} r_1 \xrightarrow{a} r_1$, and if Spoiler does $q_1 \xrightarrow{a} q_0 \xrightarrow{b} q_1$, then Duplicator does $r_1 \xrightarrow{a} r_2 \xrightarrow{b} r_1$. The other cases are similar.
- $p_0 \not\sqsubseteq^k r_0$, for any $k > 0$. From r_0, Duplicator can play a trace for any word w of length $k > 0$, but in order to extend it to a trace of length $k+1$ for any $w' = wa$ or wb, she needs to know whether the last $(k+1)$-th symbol is a or b. Thus, no finite lookahead suffices for Duplicator.

Incidentally, notice that r_0 simulates p_0 with *k*-continuous simulation, and $k = 2$ suffices.

As shown in Section 4, non-transitivity of lookahead simulation is not an obstacle to its applications. Since it is only used to compute good under-approximations of certain preorders, one can simply consider its transitive closure (which is easily computed).

C. Fixpoint Logic Characterization of Lookahead Simulation

In this section we give a fixpoint logic characterization of lookahead simulation, using the modal μ-calculus. Basically it follows from the following preservation property enjoyed by lookahead simulation: Let $x \in \{\mathrm{di}, \mathrm{de}, \mathrm{f}, \mathrm{bw}\}$ and $k > 0$. When Duplicator plays according to a winning strategy, in any configuration (p_i, q_i) of the resulting play, $p_i \sqsubseteq^{k\text{-}x} q_i$. Thus, k-lookahead simulation (without acceptance condition) can be characterized as the largest $X \subseteq Q \times Q$ which is closed under a certain monotone predecessor operator. For convenience, we take the point of view of Spoiler, and compute the complement relation $W^x = (Q \times Q) \setminus \sqsubseteq^{k\text{-}x}$ instead. This is particularly useful for delayed simulation, since we can avoid recording the obligation bit (see [14]) by using the technique of [24].

Direct and backward simulation. Consider the following predecessor operator $\mathsf{CPre}^{\mathrm{di}}(X)$, for any set $X \subseteq Q \times Q$:

$$\begin{aligned}\mathsf{CPre}^{\mathrm{di}}(X) = \{(p_0,q_0) \mid \exists (p_0 \xrightarrow{a_0} p_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{k-1}} p_k)\\ \forall (q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{m-1}} q_m), 0 < m \le k,\\ \textit{either} \quad \exists (0 \le j \le m) \cdot p_j \in F \text{ and } q_j \notin F,\\ \textit{or} \quad (p_m,q_m) \in X\}\end{aligned}$$

Intuitively, $(p,q) \in \mathsf{CPre}^{\mathrm{di}}(X)$ iff, from position (p,q), in one round of the game Spoiler can either force the game in X, or violate the winning condition for direct simulation. For backward simulation, $\mathsf{CPre}^{\mathrm{bw}}(X)$ is defined analogously, except that transitions are reversed and also initial states are taken into account:

$$\begin{aligned}\mathsf{CPre}^{\mathrm{bw}}(X) = \{(p_0,q_0) \mid \exists (p_0 \xleftarrow{a_0} p_1 \xleftarrow{a_1} \cdots \xleftarrow{a_{k-1}} p_k)\\ \forall (q_0 \xleftarrow{a_0} q_1 \xleftarrow{a_1} \cdots \xleftarrow{a_{m-1}} q_m), 0 < m \le k,\\ \textit{either} \quad \exists (0 \le j \le m) \cdot p_j \in F \text{ and } q_j \notin F,\\ \textit{or} \quad \exists (0 \le j \le m) \cdot p_j \in I \text{ and } q_j \notin I,\\ \textit{or} \quad (p_m,q_m) \in X\}\end{aligned}$$

Remark The definition of $\mathsf{CPre}^x(X)$ requires that the automaton has no deadlocks; otherwise, Spoiler would incorrectly lose if she can only perform at most $k' < k$ transitions. We assumed that the automaton is complete to keep the definition simple, but our implementation works with general automata.

Intuitively, the generalization to incomplete automata works as follows. If Spoiler's move reaches a deadlocked state after k' steps, where $1 \le k' < k$ then Spoiler does not immediately lose. Instead Duplicator needs to reply to this move of length k'. In other words, if Spoiler's move ends in a deadlocked state then the lookahead requirements are weakened, because one simply cannot demand any more steps from Spoiler.

For $X = \emptyset$, $\mathsf{CPre}^x(X)$ is the set of states from which Spoiler wins in at most one step. Thus, Spoiler wins iff she can eventually reach $\mathsf{CPre}^x(\emptyset)$. Formally, for $x \in \{\mathrm{di}, \mathrm{bw}\}$,

$$W^x = \mu W \cdot \mathsf{CPre}^x(W)$$

Delayed and fair simulation. We introduce a more elaborate three-arguments predecessor operator $\mathsf{CPre}(X,Y,Z)$. Intuitively, a configuration belongs to $\mathsf{CPre}(X,Y,Z)$ iff Spoiler can make a move s.t., for any Duplicator's reply, at least one of the following conditions holds:

1. Spoiler visits an accepting state, while Duplicator never does so; then, the game goes to X.
2. Duplicator never visits an accepting state; the game goes to Y.
3. The game goes to Z (without any further condition).

$$\begin{aligned}\mathsf{CPre}(X,Y,Z) = \{(p_0,q_0) \mid \exists (p_0 \xrightarrow{a_0} p_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{k-1}} p_k)\\ \forall (q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{m-1}} q_m) \cdot \forall (0 < m \le k) \cdot\\ \textit{either} \quad \exists (0 \le i \le m) \cdot p_i \in F, \forall (0 \le j \le m) \cdot q_j \notin F, (p_m,q_m) \in X\\ \textit{or} \quad \forall (0 \le j \le m) \cdot q_j \notin F, (p_m,q_m) \in Y\\ \textit{or} \quad (p_m,q_m) \in Z\}\end{aligned}$$

For fair simulation, Spoiler wins iff, except for finitely many rounds, she visits accepting states infinitely often while Duplicator does not visit any accepting state at all. Thus,

$$W^{\mathrm{f}} = \mu Z \cdot \nu X \cdot \mu Y \cdot \mathsf{CPre}(X,Y,Z)$$

For delayed simulation, Spoiler wins if, after finitely many rounds, the following conditions are both satisfied: 1) She can visit an accepting state, and 2) She can prevent Duplicator from visiting accepting states in the future. For condition 1), let $\mathsf{CPre}^1(X,Y) := \mathsf{CPre}(X,Y,Y)$, and, for 2), $\mathsf{CPre}^2(X,Y) := \mathsf{CPre}(X,X,Y)$. Then,

$$W^{\mathrm{de}} = \mu W \cdot \mathsf{CPre}^1(\nu X \cdot \mathsf{CPre}^2(X,W),W)$$